REPORT DOCUMENTATION PAGE	Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY) 28-10-2007	2. REPORT TYPE Final Report	3. DATES COVERED (From - To) 15-Sep-2006 - 14-Sep-2007

4. TITLE AND SUBTITLE ARO Workshop on Security of Embedded Systems and Networks	5a. CONTRACT NUMBER W911NF-06-1-0448
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 5M30V2
6. AUTHORS Peng Ning, Frank Mueller	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES North Carolina State University Office of Contract and Grants Leazar Hall Lower Level- MC Raleigh, NC 27695 -7214	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709-2211	10. SPONSOR/MONITOR'S ACRONYM(S) ARO 11. SPONSOR/MONITOR'S REPORT NUMBER(S) 51429-CI-CF.1

12. DISTRIBUTION AVAILIBILITY STATEMENT

Approved for public release; Federal purpose rights

13. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

Embedded systems and networks are used heavily in critical defense applications. Malicious or accidental failures in embedded systems can have dire consequences. The integrity of embedded software infrastructures, such as configuration and code, is of utmost importance. The autonomous nature of embedded systems also poses new challenges in the context of system integrity.

The purpose of this workshop was for a group of experts to present the state of art of embedded system and network security

15. SUBJECT TERMS

embedded systems, embedded networks, security, adversary model, software security, hardware security, robust distributed services, Languages and Software Engineering, Wireless Sensor Networks, MANET and Cellular Security

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Peng Ning
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	UU		19b. TELEPHONE NUMBER 919-513-4457

Report Title

Report on the 2007 ARO Planning Workshop on Embedded Systems and Network Security

ABSTRACT

Embedded systems and networks are used heavily in critical defense applications. Malicious or accidental failures in embedded systems can have dire consequences. The integrity of embedded software infrastructures, such as configuration and code, is of utmost importance. The autonomous nature of embedded systems also poses new challenges in the context of system integrity.

The purpose of this workshop was for a group of experts to present the state of art of embedded system and network security research and to discuss and develop a research roadmap. Each invitee contributed a position paper/statement and had 20 minutes to present his/her research interests/project. A 30-minute group discussion / Q&A session was also scheduled at the end of each session. The research areas covered under the workshop included

1. Adversary models
2. Languages and Software Engineering
3. Software Security
4. Hardware Security
5. Robust Distributed Systems
6. Wireless Sensor networks, MANET and Cellular Security

This report gives the position papers/statements contributed by the workshop participants, the slides used during the workshop, the summary of each session, and the research challenges identified by the workshop participants in embedded systems and network security.

List of papers submitted or published that acknowledge ARO support during this reporting period. List the papers, including journal references, in the following categories:

(a) Papers published in peer-reviewed journals (N/A for none)

Number of Papers published in peer-reviewed journals: 0.00

(b) Papers published in non-peer-reviewed journals or in conference proceedings (N/A for none)

Number of Papers published in non peer-reviewed journals: 0.00

(c) Presentations

Number of Presentations: 0.00

Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

Number of Non Peer-Reviewed Conference Proceeding publications (other than abstracts): 0

Peer-Reviewed Conference Proceeding publications (other than abstracts):

Number of Peer-Reviewed Conference Proceeding publications (other than abstracts): 0

(d) Manuscripts

Number of Manuscripts: 0.00

Number of Inventions:

Graduate Students

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Names of Post Doctorates

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Names of Faculty Supported

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Names of Under Graduate students supported

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Student Metrics

This section only applies to graduating undergraduates supported by this agreement in this reporting period

The number of undergraduates funded by this agreement who graduated during this period:	0.00
The number of undergraduates funded by this agreement who graduated during this period with a degree in science, mathematics, engineering, or technology fields:......	0.00
The number of undergraduates funded by your agreement who graduated during this period and will continue to pursue a graduate or Ph.D. degree in science, mathematics, engineering, or technology fields:......	0.00
Number of graduating undergraduates who achieved a 3.5 GPA to 4.0 (4.0 max scale):......	0.00
Number of graduating undergraduates funded by a DoD funded Center of Excellence grant for Education, Research and Engineering:......	0.00
The number of undergraduates funded by your agreement who graduated during this period and intend to work for the Department of Defense	0.00
The number of undergraduates funded by your agreement who graduated during this period and will receive scholarships or fellowships for further studies in science, mathematics, engineering or technology fields:......	0.00

Names of Personnel receiving masters degrees

NAME

Total Number:

Names of personnel receiving PHDs

NAME

Total Number:

Names of other research staff

NAME	PERCENT_SUPPORTED

FTE Equivalent:

Total Number:

Sub Contractors (DD882)

Inventions (DD882)

Proceedings of 2007 ARO Planning Workshop on Embedded Systems and Network Security

February 22-23, 2007

Raleigh, NC, USA

Edited by Peng Ning

ARO Planning Workshop on Embedded Systems and Network Security

February 22-23, 2007, Raleigh, NC, USA

Workshop Organizers:

Cliff Wang (ARO)
Peng Ning (NCSU)
Frank Mueller (NCSU)

Workshop Participants:

Kang Shin (U Mich)
John Stankovic (UVA)
Phil Koopman (CMU)
Wenliang Du (Syracuse U.)
Virgil Gligor (UMD)
Radha Poovendran (UW)
Adrian Perrig (CMU)
Gene Tsudik (UCI)
Mike Reiter (CMU)
Jack Davidson UVA)
Hsien-Hsin Lee (GaTech)
Suman Banerjee (U Wisc.)
Somesh Jha (U Wisc.)
Sean Smith (Dartmouth U.)
Wenye Wang (NCSU)
Purush Iyer (NCSU)
Yan Solihin (NCSU)
Tao Xie (NCSU)

Background

Embedded systems and networks are used heavily in critical defense applications. Malicious or accidental failures in embedded systems can have dire consequences. The integrity of embedded software infrastructures, such as configuration and code, is of utmost importance. The autonomous nature of embedded systems also poses new challenges in the context of system integrity. Since embedded systems are reactive, unexpected environment events or environment events generated by a malicious adversary can cause failures in embedded systems and networks. Embedded systems and networks often have to operate autonomously in a changing environment. Therefore, infrastructure of an embedded system has to be updated to adapt its behavior to the change in environment or the overall mission. Unauthorized or unverified updates to the infrastructure of an embedded system can compromise its integrity. This workshop intends to bring researchers that have expertise in a variety of techniques for ensuring the security and integrity of mission-critical embedded systems and networks.

Objective of the Workshop

The purpose of this workshop was for a group of experts to present the state of art of embedded system and network security research and to discuss and develop a research roadmap. Each invitee contributed a position paper/statement and had 20 minutes to present his/her research interests/project. A 30-minute group discussion / Q&A session was also scheduled at the end of each session.

The research areas covered under the workshop included

1. Adversary models
2. Languages and Software Engineering
3. Software Security
4. Hardware Security
5. Robust Distributed Systems
6. Wireless Sensor networks, MANET and Cellular Security

The following sections of this report give the position papers/statements contributed by the workshop participants, the slides used during the workshop, the summary of each session, and the research challenges identified by the workshop participants in embedded systems and network security.

Session I – Adversary Models

Session Chair: Peng Ning

Handling New Adversaries in Secure Mobile Ad-hoc Networks

Virgil D. Gligor
Electrical and Computer Engineering Department
University of Maryland
College Park, Maryland 20742

1. The Problem

Invariably, new technologies introduce new vulnerabilities which often enable new attacks by increasingly potent adversaries. Yet new systems are more adept at handling well-known attacks by old adversaries than anticipating new ones. Our adversary models seem to be perpetually out of date: often they do not capture adversary attacks and sometimes they address attacks rendered impractical by new technologies. An immediate consequence of using an out-of-date adversary model with a new technology is that security analysis methods and tools cannot possibly handle the new vulnerabilities thereby leaving users exposed to new attacks. An equally compelling reason for investigating new adversarial capabilities in Mobile Ad-hoc Networks (MANETS) is this: without a precise adversary definition the very notion of security becomes undefined. For instance, the fundamental question of "what is the set of threats addressed" by a given secure protocol cannot be answered without an adversary definition.

In short, we need to provide (1) a new definition for the new adversary attacks made possible by Mobile Ad-hoc Networks (MANETS), (2) demonstrate that this new definition is more general than the traditional, formal network adversary models (including the classic Dolev-Yao and Byzantine models), (3) illustrate how this new adversary is countered with new practical protocols that operate under realistic performance and cost constraints. Interesting protocols to investigate using the new adversarial definition include those typically used in MANET management, distributed sensing and data fusion, as well as the more traditional authentication protocols for principal and node-to-node authentication.

2. Background

A common vulnerability of MANETS, and in general of all networks whose nodes operate in hostile environments, is is the possibility of physical capture and control of network devices by an adversary. Frank Stajano's "big stick principle," which states that whoever has physical control of a device is allowed to take it over, suggests that such an adversary is "difficult" to counter. In fact, no amount of device protection, nor increased computational workload imposed on this adversary, seems to suffice: the adversary can selectively control the inputs to network devices without causing physical tampering and thus can corrupt network operations, and can selectively jam the outputs of network devices in a stealthy manner and thus deny network operations. This implies that protecting device secrets (e.g., cryptographic keys) via physical security measures, which currently range from those employed by smartcards (very little tamper resistance) to those of IBM 4758 crypto co-processors (highest FIPS 140 evaluation), is both unrealistic and inadequate in the face of the new adversary. Even when the cost of strong physical security measures is affordable in some traditional networking environments (e.g., banking), such protection is inadequate in MANETS because access to a node's internal state is (1) usually possible without direct access to the protected cryptographic keys and (2) typically the form factors and resource requirements (e.g., energy) of the protective devices (e.g., IBM 4758 card) are not suitable for the limited power and small form-factor MANET nodes. Thus, in captured MANET nodes (e.g., PDAs, laptops) access to the internal states by an adversary cannot always be prevented.

A further problem caused by this new and "difficult" adversary is that of adaptive capture of MANET nodes: once a node is physically captured and its internal state discovered, all the secrets (e.g., cryptographic keys) which the node may use for authentication with with other nodes are compromised. Now the adversary can proceed to selectively capture additional nodes that execute network applications. Thus the new adversary can control multiple nodes of a network thereby enabling *collusion attacks* perpetrated by cooperating captured nodes.

A new MANET adversary model should include new features that are currently not present in the traditional formal models. To see this, let us recall, for instance, the key features of the Dolev-Yao model that dominated most analysis of cryptographic protocols for the past two decades. The Dolev-Yao model has three basic components, namely:

1) the presence of the the "man-in-the-middle" (MITM) everywhere in the network. That is, the adversary can launch any MITM attack on any and all network links and thus can read, replay, block, insert messages anywhere.

2) the adversary can send and receive messages from any legitimate principal (e.g., node) of the network. Thus, the adversary can freely communicate with all legitimate principals and nodes of the network and use them as oracles in attempts to discover secrets and forge messages. And,

(3) the adversary can be a legitimately registered principal of the network. Thus, s/he can attack other network nodes by exploiting protocol features and vulnerabilities.

While the Dolev-Yao adversaries appear to be extremely powerful in any network, they lack the capabilities of the new network adversary enabled by MANETS. For instance, the Dolev-Yao adversary cannot capture network nodes an discover other principals' or other nodes' secrets. Further, this adversary does not address the threat of *collusion attacks* launched by cooperating captured nodes under the adversary's control. Finally, this adversary cannot modify a network's trust and physical topology. For instance, a Dolev-Yao adversary cannot read a node's internal state, replicate it on other nodes under its control and insert the controled nodes within the network.

A similar analysis shows that the traditional Byzantine adversaries typically used in consensus protocols are also less general than the new MANET adversaries. For example, such adversaries have a "threshold" behavior: below a fixed threshold of captured nodes they can be countered (e.g., 1/3 captured nodes if message authentication cannot be provided and a simple minority, otherwise). In MANETS applications, substantial damage can be perpetrated even by capturing substantially fewer nodes than the Byzantine thresholds indicate. Further, the traditional notion of adversary "mobility," which suggests that the Byzantine adversary captures a set of up to "t" nodes in some protocol state and then captures a totally different set of up to "t" nodes in another state [5], has changed. The new adversary's behaviour is monotonic and not limited to "t" nodes: once a node is captured, it stays that way and the number of captured nodes is not limited to a fixed threshold value, "t."

3. What is Needed ?

We suggest that an adversary model is needed that is suitable for the new threats posed by using MANET technologies in hostile environments. Once a comprehensive definition of the adversary is given, it become necessary to investigate how this adversary can be handled in practical ways within the preformance and cost constraints of typical MANETS. Specifically we nned to investigate how to handle the new adversary within specific MANET protocols.

While perfect physical security of ad-hoc network devices is both currently unrealistic and fundamentally inadequate a goal, "good-enough" network security in the face of "difficult" MANET adversaries can be obtained with relatively inexpensive technologies. For example, algorithmic adversary-detection technologies can be based on *emergent properties and protocols*. Intuitively, emergent properties are features that cannot be provided by individual network nodes themselves but instead result from interaction and collaboration among these nodes. Although one may think of the creation of an ad-hoc network as a set of emergent connectivity and routing properties, our primary focus is on the specific properties that may emerge after the ad-hoc networks are thus established. The emergent properties and protocols we propose to study for the handling of "difficult" MANET adversaries are different from traditional network properties established via protocol interactions in several fundamental ways. First, it is possible that neither the time nor the locus of emergence of these properties can be easily anticipated. Second, the emergence of these properties may be uncertain, in the sense that it may be probabilistic. Third, these properties may be transient, in the sense that they may disappear from the ad-hoc network during normal operation and not as a result of exceptional events; e.g., node or protocol failures.

We believe emergent properties and protocols are essential to handling ”difficult” adversaries; e.g., adversaries that exceed the powers of the traditional ”Dolev-Yao” and ”Byzantine” adversaries. Emergent properties can be used to detect, often probabilistically, the presence of a ”difficult” adversary and to pinpoint with reasonable accuracy the affected network area (e.g., identify a specific captured node, a particular property of captured nodes) [6, 4]. Correct assessment of node capture and replication is important for otherwise false detection may also lead to node revocation [2], which in turn may lead to network partitioning and denial of service. Similarly missed detection may lead to node replication and collusion among replicas also leading to network partitioning and/or false data injection and application corruption.

Finally, emergent properties help determine the scalability and resilience of ad-hoc networks. For example, emergent properties (such as establishment of secure communication paths in sensor networks via random key pre-distribution) may place constraints on the network size but may also imply resilience of network communications below a certain threshold of compromised nodes [1].

References

[1] H. Chan, A. Perrig, and D. Song, “Random Key Predistribution Schemes for Sensor Networks,” Proc. of the IEEE Security and Privacy Symposium, Berkeley, CA, May 2003 (available at *http://www.ece.cmu.edu/˜adrian*).

[2] H. Chan, V. Gligor, A. Perrig, and G. Muralidharan, ”On the Distribution and Revocation of Cryptographic Keys in Sensor Networks,” *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 3, July-Sept. 2005.

[3] L. Eschenauer, V.D Gligor, and J.S. Baras, “On Trust Establishment in Mobile Ad-Hoc Networks”, in *Security Protocols*, Christianson *et al.* (eds.), Cambridge, UK, April 2002. (available at *http://www.ee.umd.edu/˜gligor*)

[4] J. McCune, E. Shi, A. Perrig and M. K. Reiter. “Detection of Denial-of-Message Attacks on Sensor Network Broadcasts,” Proc. of the IEEE Symp. on Security and Privacy, Oakland, California 2005

[5] R. Ostrovsky and M. Yung, “How to Withstand Mobile Virus Attacks,” ACM Symp. on Principles of Distributed Computing, 1991, pp. 51-59.

[6] B. Parno, A. Perrig, V. Gligor “Distributed Detection of Node Replication Attacks in Sensor Networks,” IEEE Symposium on Security and Privacy, Oakland, CA. 2005. (available at *http://www.ece.cmu.edu/˜adrian*).

Some challenges in wireless security

Suman Banerjee
Department of Computer Sciences, University of Wisconsin, Madison, WI 53706, USA
Email: suman@cs.wisc.edu

1 Introduction

Wireless communication technologies provide users with significant flexibility and portability and hence is being widely adopted as a preferred mode of communication in many military and civilian applications. By eliminating the need for devices to be tethered by wires, such technologies enable new usage scenarios not otherwise possible. A number of mobile, in-range wireless devices can self-organize themselves into an ad-hoc network — such capabilities have many applications, e.g., for first responders.

However, such increased flexibility comes with increased vulnerabilities. Many unique vulnerabilities in the wireless environment occur due to the shared and open nature of communication. In this short document, we discuss some of these new threats, recent approaches to mitigate them, and further challenges that need to be addressed.

2 Potential threats

While many of the vulnerabilities in wireless environments are similar to those in a wired network, others are fairly unique in nature. A lot of ongoing research and design efforts are addressing many of these vulnerabilities. Some of these issues that have received significant attention in recent years include mutual authentication to users prior to communication, as well as data confidentiality, and data integrity. For example, the 802.11i standards are being used to provide such authentication in wireless LAN environments [1]. However, even if these concerns are reasonably addressed, many further challenging security concerns remain.

- Availability attacks: The goal of these attacks are to reduce the availability of the wireless medium to legitimate users. The inherent broadcast nature of the wireless medium implies that an attacker can easily mount such attacks by selectively interfering with legitimate communication. For example, an attacker can transmit packets with high NAV values, that prevent any legitimate user from accessing the channel for long durations of time.

- Energy attacks: An attacker can easily send wireless traffic to a victim node that requires the latter to process such traffic prior to realizing such traffic to have no local relevance. However, the effort of decoding such traffic requires power consumption, and slowly drains the battery of the mobile node. Such attacks are easy to mount and require sophisticated strategies to combat.

- Location privacy and authentication: An attacker can monitor communication patterns in the medium to continuously track the location of different users in the environment. In order to guard against such attack capabilities, it is important to design strategies that allow users privacy of their location information. However, design of such tools can benefit attackers too — if location privacy can be carefully preserved, then attackers can utilize them to hide their own location from the system.

3 Approaches to mitigate such attacks

We now discuss some approaches that may be useful in mitigating such attacks in wireless environments.

- Availability attacks: The simplest form of an availability attack is PHY layer ’bit-jamming’ approach, where the attacker causes continuous interference on the medium. Such attacks, however, are energy inefficient. They require attackers to expend a lot of energy, a process which can potentially make the attack detection and mitigation tasks easier. In contrast, an intelligent attacker may choose to attack the MAC layer of the protocol stack. Since MAC layers typically define cooperation-based techniques for medium access, an attacker (a non-cooperating node) can selectively mount attacks on specific MAC control frames to disrupt all MAC level communication. Such an attack incurs very low energy costs at the attacker.

Hence, mitigation of availability attacks both at PHY and MAC layers should be an important component of the wireless security research agenda. At the PHY layer, number of interesting approaches can be researched, including various channel-matched signaling schemes, efficient packet coding, on top of conventional spread-spectrum and antenna nulling approaches. At the MAC layer, it may be possible to design new MAC protocols that utilize randomization and obfuscation techniques making it difficult for attackers to opportunistically attack or fake MAC control frames.

- Energy attacks: These attacks are quite simple to mount and yet very effective in disrupting communication. Within the context of this workshop, these attacks are are applicable to both MANETs and mobile phones. For mobile phones this problem can be particularly vexing, since the user has no explicit way of pre-filtering against such irrelevant messages sent by an attacker. We believe that proper security against these attacks would be multi-dimensional. One component of the solution will include support from less energy constrained devices that are not necessarily co-located with the device under attack. Another approach may be to design strategies that can quickly evaluate the value of incoming packets to the user. However, the speed (and energy costs) of such evaluation would trade-off against the accuracy of the decision process.

- Location privacy and authentication: There is an inherent tradeoff between the privacy of location information of users and the ability for the system to authenticate the same information. The biggest challenge in maintaining location privacy stems from the ability of an attacker to triangulate the location of a transmitter based on received signal strength, angle or arrival, and other such properties. Different strategies can be considered to provide location privacy both at PHY and MAC layers. For example, it might be possible to utilize antenna-nulling techniques that obfuscate signal strength information. Similarly, the system may design techniques to induce additional interference that achieves the same effect.

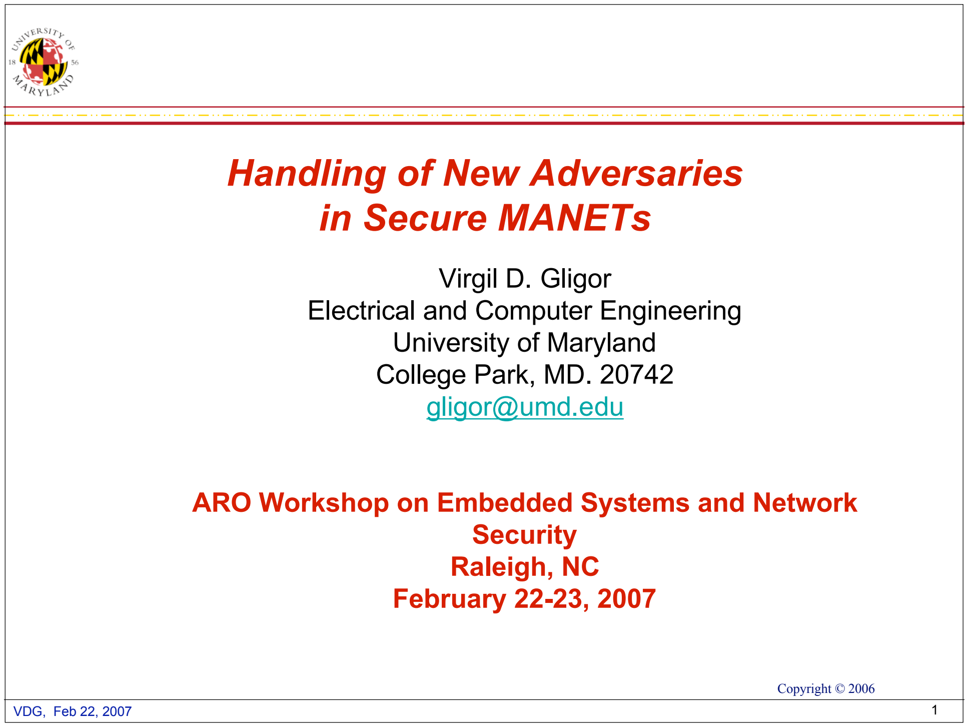
Location authentication is also a difficult problem because tools useful for privacy can be employed by attackers to guard against the system's ability to determine the user's or the attacker's location. Past research has studied different statistical techniques for location determination that are based on received signal strength from different transmitters at a given location. However, location determination based on received signal strength information is relatively easy to attack. In particular, an attacker can construct efficient models that allows it to infer received signal strength in different parts of the physical space. Such a capability will allow an attacker to fake its own location to the system. Some recent work utilizes the notion of wireless congruity [2], which provides location authentication based on the "common experience" of nodes that are physically close to each other. Hence, if the location of a set of trusted reference points can be determined, then a proximity metric can be defined that allows for location authentication. Further evaluation of these and other such schemes need to studied.

Finally, the tradeoffs between location privacy mechanisms and the need for location authentication is a challenging domain of work and will be an important direction of future study.

References

[1] IEEE. Amendment to standard for telecommunications and information exchange between systems - LAN/MAN specific requirements - part 11: Wireless medium access control (MAC) and physical layer (PHY) specifications: Medium access control (MAC) security enhancements. IEEE Standard 802.11i, 2004.

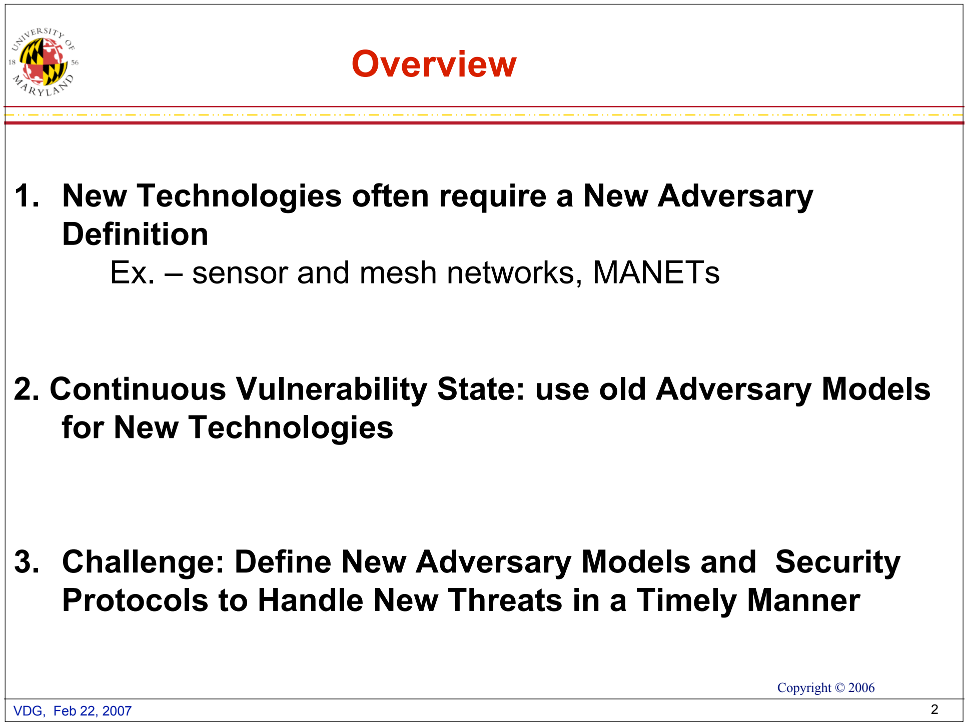
[2] A. Mishra, S. Rayanchu, A. Shukla, and S. Banerjee. Towards robust localization using wireless congruity. In *ACM HotMobile*, February 2007.



Handling of New Adversaries in Secure MANETs

Virgil D. Gligor
Electrical and Computer Engineering
University of Maryland
College Park, MD. 20742
gligor@umd.edu

ARO Workshop on Embedded Systems and Network Security
Raleigh, NC
February 22-23, 2007



Overview

1. **New Technologies often require a New Adversary Definition**
   Ex. – sensor and mesh networks, MANETs

2. **Continuous Vulnerability State: use old Adversary Models for New Technologies**

3. **Challenge: Define New Adversary Models and Security Protocols to Handle New Threats in a Timely Manner**

A system without an adversary definition cannot possibly be insecure; it can only be astonishing…

… astonishment is a much underrated security vice.

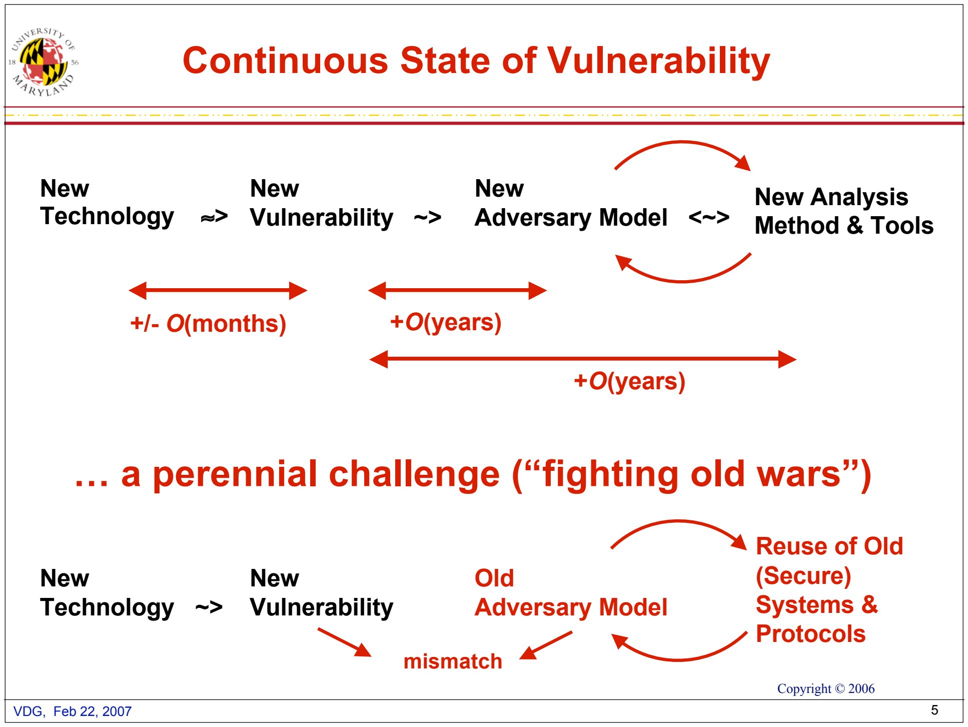
(Principle of Least Astonishment)

Why an Adv. Def. is a fundamental concern ?

1. New Technology ≈> Vulnerability ~> Adversary <~> Methods & Tools

-sharing user-mode programs& data; - computing utility (early – mid 1960s)	confidentiality and integrity breaches; system penetration;	untrusted user-mode programs & subsystems	sys. vs. user mode ('62->) rings, sec. kernel ('65, '72) FHM ('75) theory/tool ('91)* acc. policy models ('71)
- shared *stateful* services e.g., DBMS, net. protocols dyn. resource alloc. (early - mid 1970s)	DoS instances	untrusted user processes; concurrent, coord. attacks	DoS = a diff. prob.(83-'85)* formal spec. & verif. ('88)* DoS models ('92 ->)
- PCs, LANs; public-domain Crypto (mid 1970s)	read, modify, block, replay, forge messages	"man in the middle" active, adaptive network adversary	informal: NS, DS ('78–81) semi-formal: DY ('83) Byzantine ('82 –>) crypto attk models ('84->) auth. prot. analysis (87->)
- internetworking (mid – late 1980s)	large-scale effects: worms, viruses, DDoS (e.g., flooding)	geo. distributed, coordinated attacks	virus scans, tracebacks intrusion detection (mid '90s ->)

2. Technology Cost -> 0, Security Concerns persist



Continuous State of Vulnerability

New Technology ≈> New Vulnerability ~> New Adversary Model <~> New Analysis Method & Tools

+/- *O*(months)

+*O*(years)

+*O*(years)

… a perennial challenge (“fighting old wars”)

New Technology ~> New Vulnerability

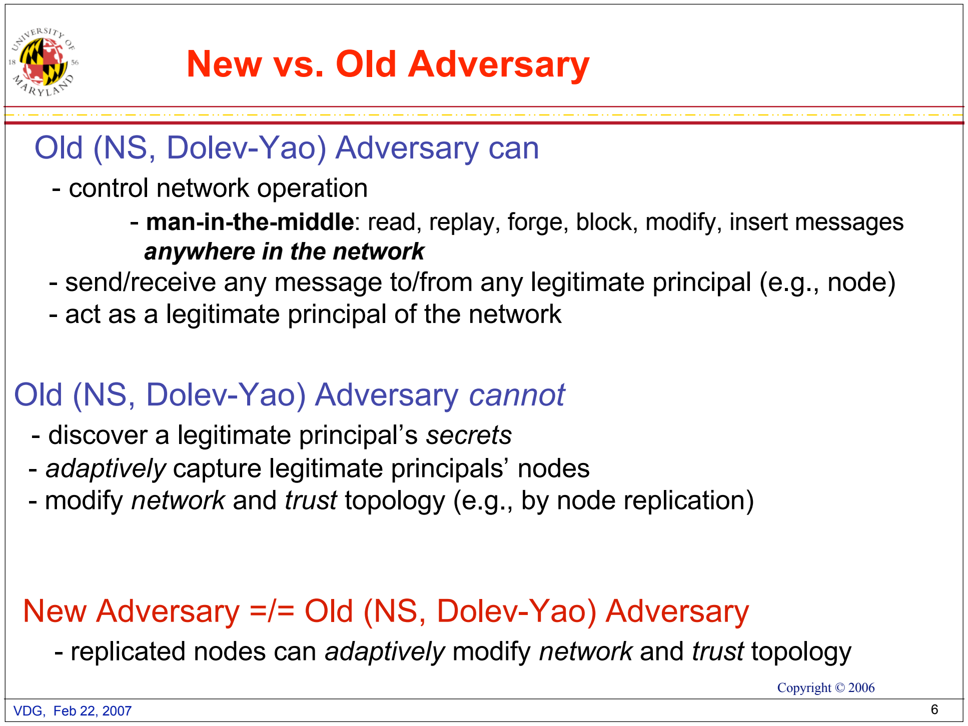
Old Adversary Model

Reuse of Old (Secure) Systems & Protocols

mismatch

VDG, Feb 22, 2007

5



New vs. Old Adversary

Old (NS, Dolev-Yao) Adversary can

- control network operation
  - **man-in-the-middle**: read, replay, forge, block, modify, insert messages ***anywhere in the network***
- send/receive any message to/from any legitimate principal (e.g., node)
- act as a legitimate principal of the network

Old (NS, Dolev-Yao) Adversary *cannot*

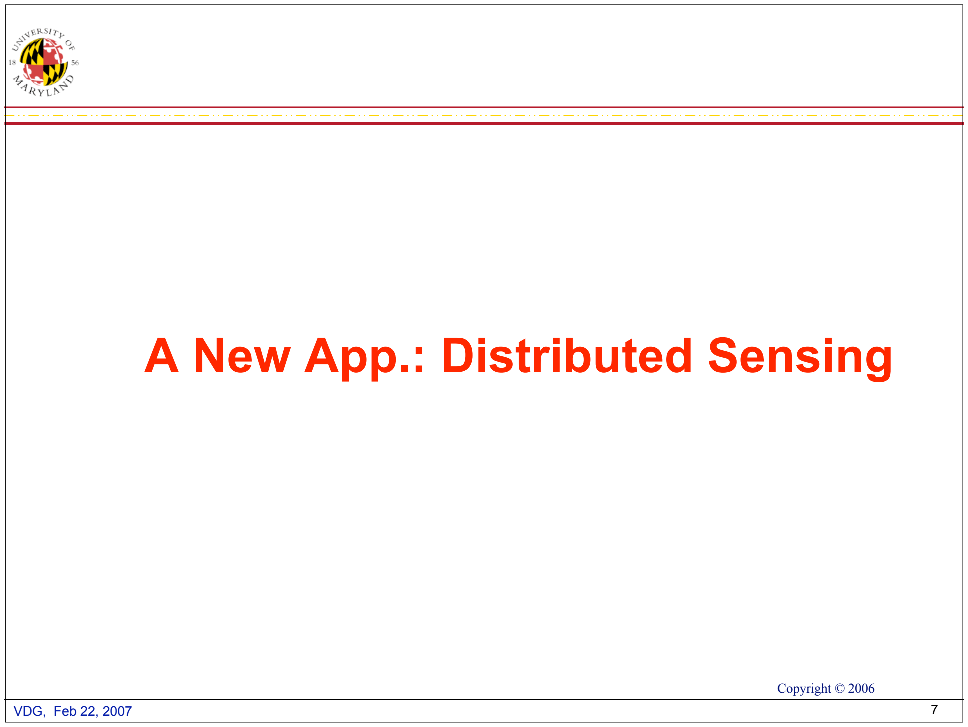
- discover a legitimate principal’s *secrets*
- *adaptively* capture legitimate principals’ nodes
- modify *network* and *trust* topology (e.g., by node replication)

New Adversary =/= Old (NS, Dolev-Yao) Adversary

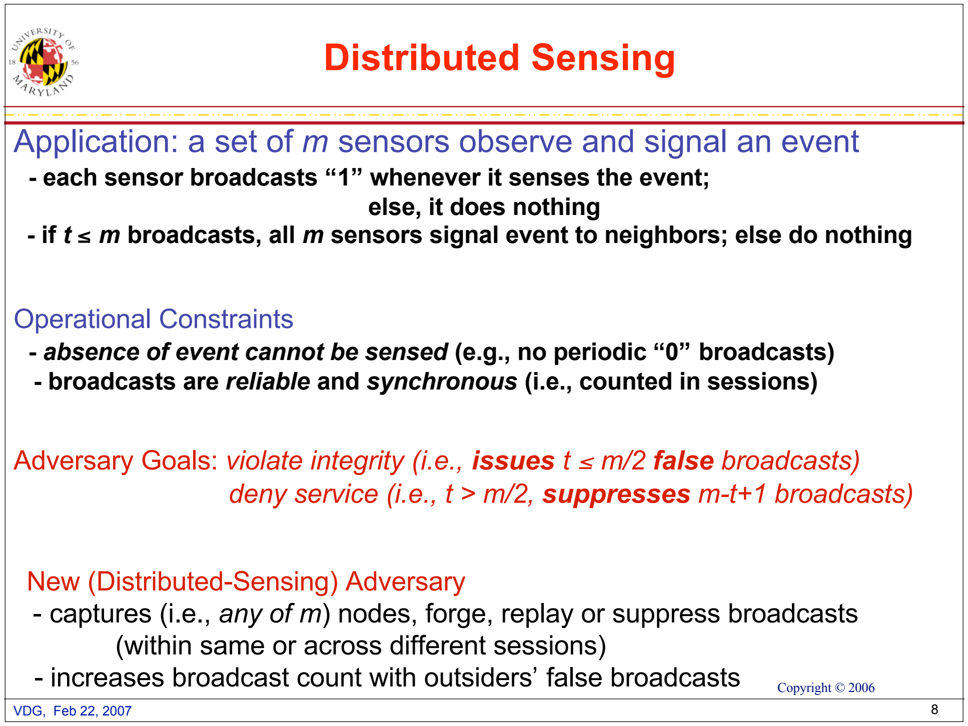
- replicated nodes can *adaptively* modify *network* and *trust* topology

VDG, Feb 22, 2007

6



A New App.: Distributed Sensing



Distributed Sensing

Application: a set of *m* sensors observe and signal an event

- each sensor broadcasts “1” whenever it senses the event;
else, it does nothing

- if *t* ≤ *m* broadcasts, all *m* sensors signal event to neighbors; else do nothing

Operational Constraints

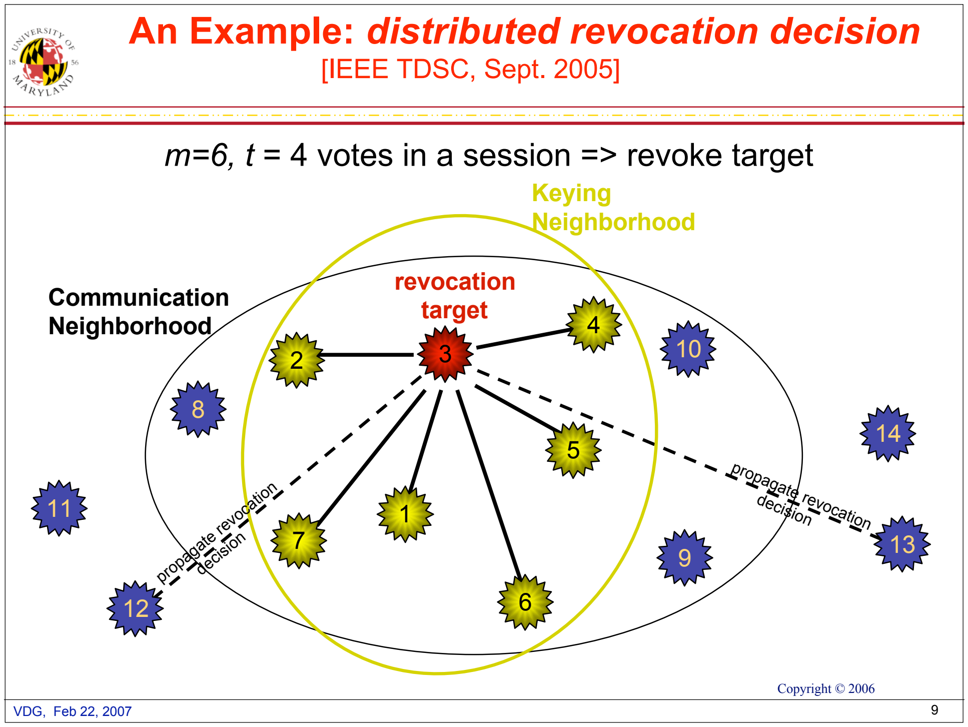
- *absence of event cannot be sensed* (e.g., no periodic “0” broadcasts)

- broadcasts are *reliable* and *synchronous* (i.e., counted in sessions)

Adversary Goals: *violate integrity (i.e., **issues** t ≤ m/2 **false** broadcasts)*
*deny service (i.e., t > m/2, **suppresses** m-t+1 broadcasts)*

New (Distributed-Sensing) Adversary

- captures (i.e., *any of m*) nodes, forge, replay or suppress broadcasts (within same or across different sessions)
- increases broadcast count with outsiders’ false broadcasts

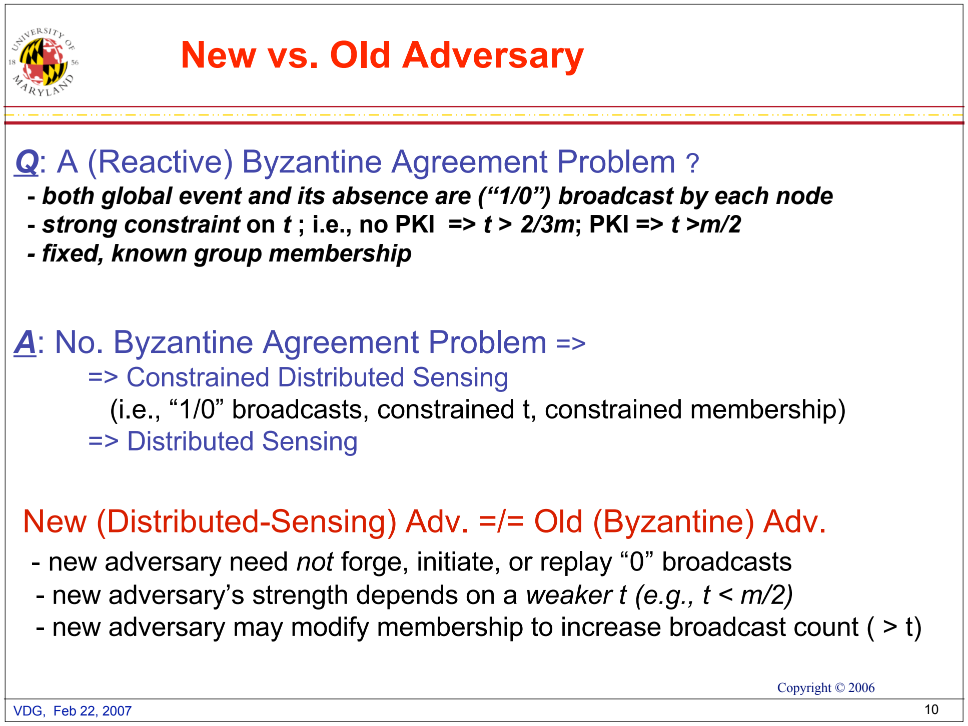


An Example: *distributed revocation decision*

[IEEE TDSC, Sept. 2005]

m=6, *t* = 4 votes in a session => revoke target

Keying Neighborhood
Communication Neighborhood
revocation target
1
2
3
4
5
6
7
8
9
10
11
12
13
14
propagate revocation decision
propagate revocation decision



New vs. Old Adversary

Q: A (Reactive) Byzantine Agreement Problem ?

- ***both global event and its absence are ("1/0") broadcast by each node***
- ***strong constraint* on *t* ; i.e., no PKI => *t* > *2/3m*; PKI => *t* >*m/2***
- ***fixed, known group membership***

A: No. Byzantine Agreement Problem =>

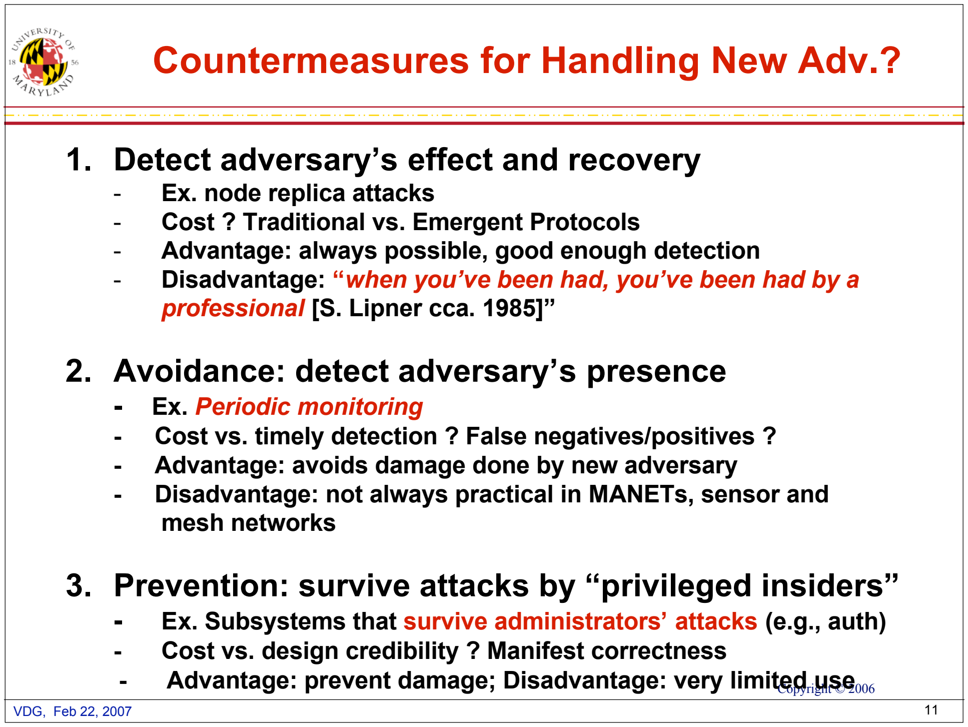
=> Constrained Distributed Sensing

(i.e., "1/0" broadcasts, constrained t, constrained membership)

=> Distributed Sensing

New (Distributed-Sensing) Adv. =/= Old (Byzantine) Adv.

- new adversary need *not* forge, initiate, or replay "0" broadcasts
- new adversary's strength depends on a *weaker t (e.g., t < m/2)*
- new adversary may modify membership to increase broadcast count (> t)

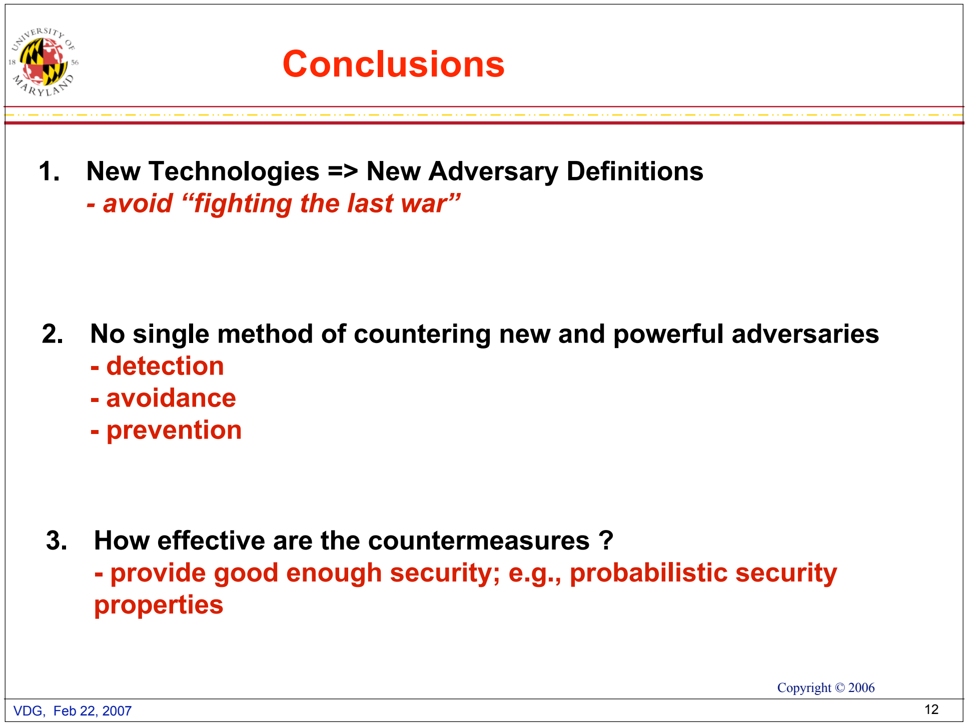


Countermeasures for Handling New Adv.?

1. Detect adversary's effect and recovery
   - Ex. node replica attacks
   - Cost ? Traditional vs. Emergent Protocols
   - Advantage: always possible, good enough detection
   - Disadvantage: *"when you've been had, you've been had by a professional* [S. Lipner cca. 1985]"

2. Avoidance: detect adversary's presence
   - Ex. *Periodic monitoring*
   - Cost vs. timely detection ? False negatives/positives ?
   - Advantage: avoids damage done by new adversary
   - Disadvantage: not always practical in MANETs, sensor and mesh networks

3. Prevention: survive attacks by "privileged insiders"
   - Ex. Subsystems that survive administrators' attacks (e.g., auth)
   - Cost vs. design credibility ? Manifest correctness
   - Advantage: prevent damage; Disadvantage: very limited use



Conclusions

1. New Technologies => New Adversary Definitions
   - avoid "fighting the last war"

2. No single method of countering new and powerful adversaries
   - detection
   - avoidance
   - prevention

3. How effective are the countermeasures ?
   - provide good enough security; e.g., probabilistic security properties

Adversary Models in Wireless Networks: Research Challenges

Radha Poovendran

Network Security Lab (NSL)

University of Washington

Questions Posed by the Committee

- What are the three fundamental limitations of today's security mechanisms?
- What are the three most important research challenges?
- What are promising innovations and abstractions for future systems?
- What are possible milestones for the next 5 to10 years?

Three fundamental limitations of today's security mechanisms

- Refer to Virgil’s presentation
- Force-fitting the old models into the wireless environment
- Considering security as an overlay instead of a critical robustness requirement
- Optimizing network performance independently of security

Most Important Challenges

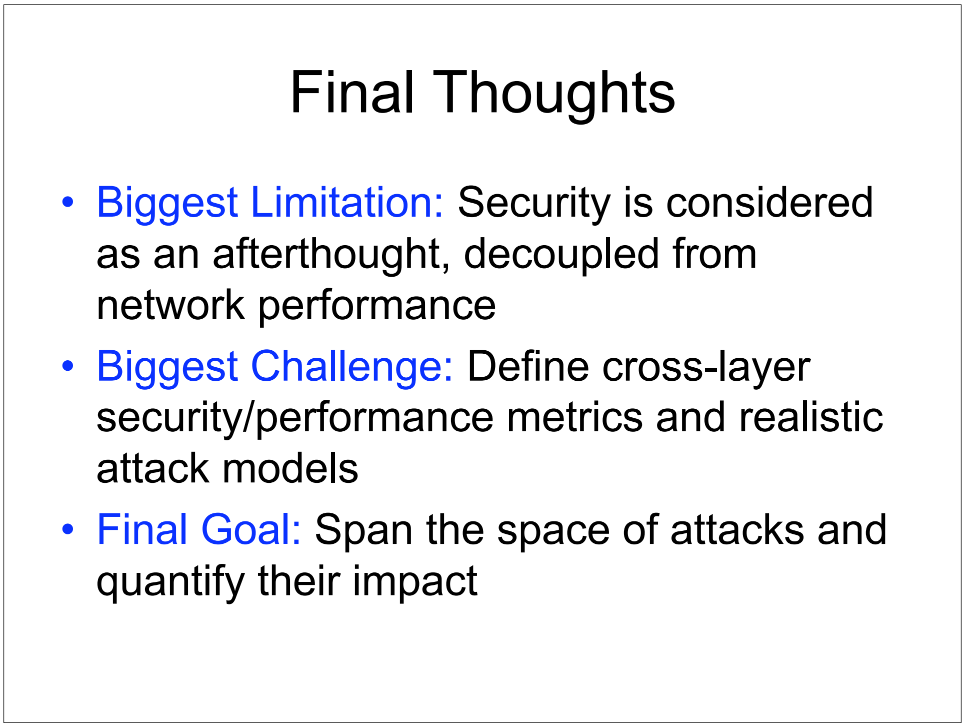
- Identifying the primitives that can be used to characterize adversary models
  - Characterize space of attacks against the network operations
  - Incorporate resource constraints for the adversary (mobility, computation, stealthiness, multiple presence)
  - Address adaptive (intelligent) as well as mobile adversary
  - Differentiate selfish vs. malicious behavior and network faults
- Defining suitable security metrics
  - To quantify the impact of an attack on the network or individual nodes
  - To couple the network performance with security
  - Flexible enough to incorporate cross-layer impact while being adaptive to attacks
- Not Ignoring the fact that in large scale networks, statistics often beat out carefully designed attacks (such as MITM)—Leading to “Passive attacks of probabilistic nature may be resource and computationally efficient than active attacks in WSN/RFID.”
- Designing security protocols that leak minimal side information!

Promising innovations and abstractions

- Graph abstractions
  - Network connectivity, Throughput
  - Robustness to intelligent attacks
- Probabilistic Analysis Techniques for
  - Modeling attacks
  - Quantifying the impact of attacks
  - Tuning defense strategy
- Potential New Primitives (More from Peng, Adrian)
- New Approaches in Network Optimization

Possible milestones for the next 5 to10 years

- Joint design of performance and security
- Development of performance metrics
  - Characterizing/Knowing the limitations of our solutions
- Adversarial models and extensions of them for
  - Heterogeneous environments
  - Resource as well as location adaptive attacks
- Also need breakthrough in new crypto primitives
  - Lightweight, suitable for resource constrained devices



Final Thoughts

- Biggest Limitation: Security is considered as an afterthought, decoupled from network performance
- Biggest Challenge: Define cross-layer security/performance metrics and realistic attack models
- Final Goal: Span the space of attacks and quantify their impact

Adversary models in wireless security

Suman Banerjee
Department of Computer Sciences
suman@cs.wisc.edu

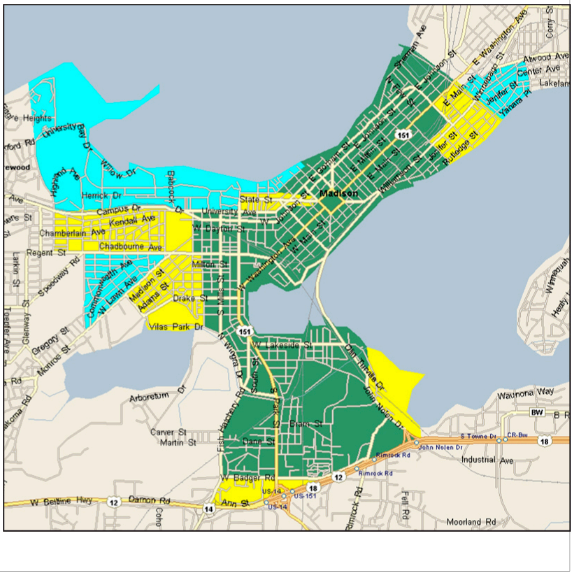
THE UNIVERSITY of WISCONSIN MADISON

Wisconsin Wireless and NetworkinG Systems (WiNGS) Laboratory

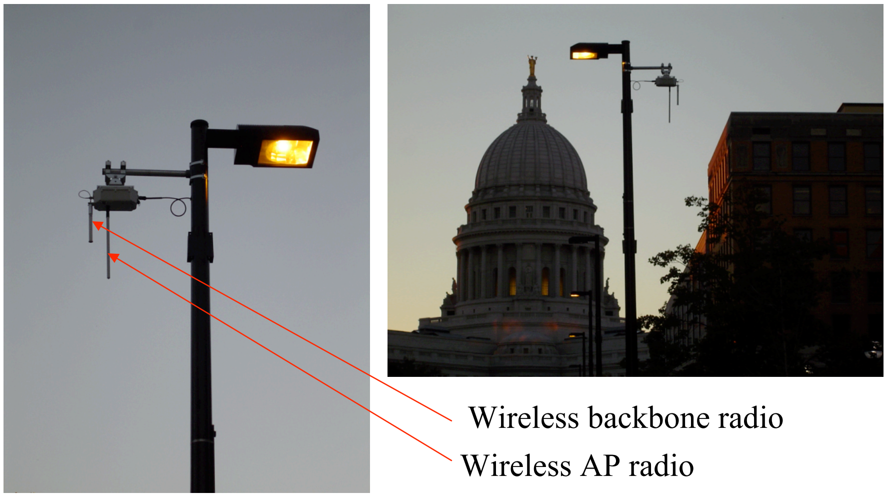
Wireless localization

Madison municipal WiFi mesh network

-
- 9 square miles area
- 200+ APs



Municipal Wi-Fi Mesh in Madison

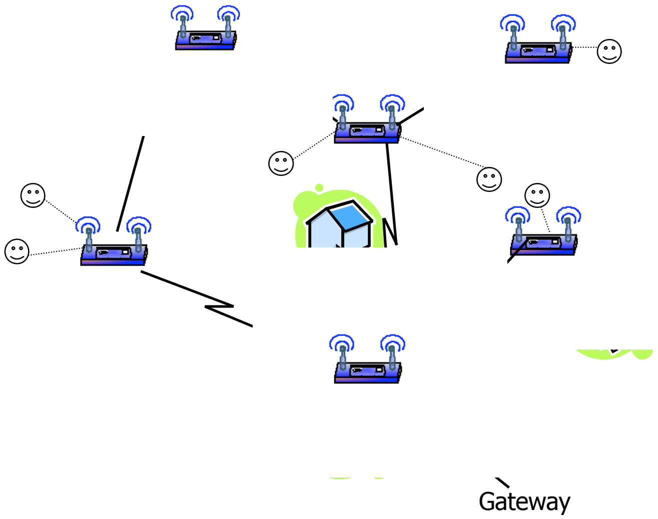


Wireless backbone radio

Wireless AP radio

Mesh AP on street light

Municipal Wi-Fi Mesh in Madison

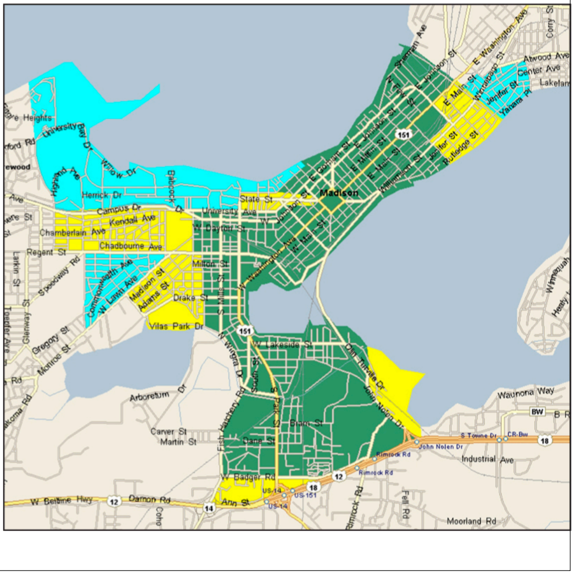


Location applications

•Assume a disaster scenario

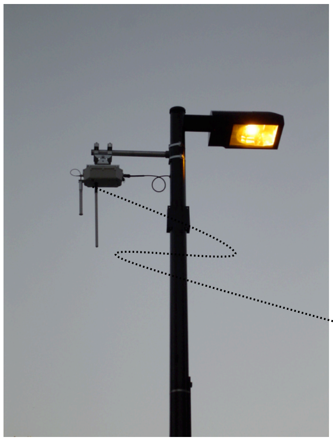
Locate position of each rescue personnel within the city in a reliable, secure fashion

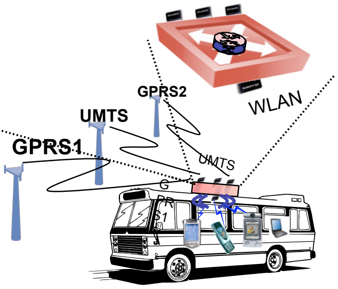
Can take advantage of existing (trusted?) WiFi mesh deployment and wireless communication of rescue personnel



Location applications

- Real-time city-bus fleet management
- Where are the different buses?



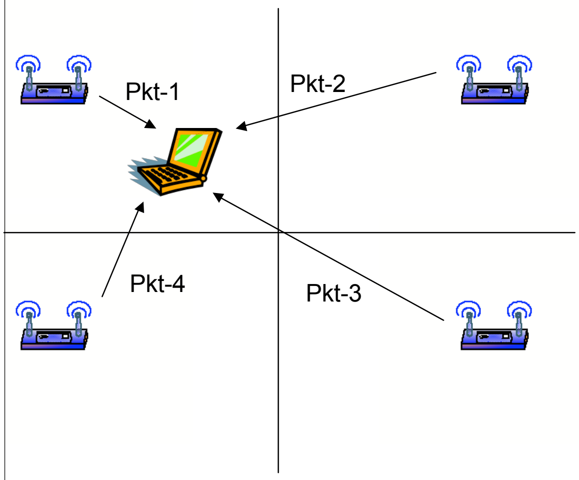


Location security

- Prove a user's location to the infrastructure
- GPS does not help

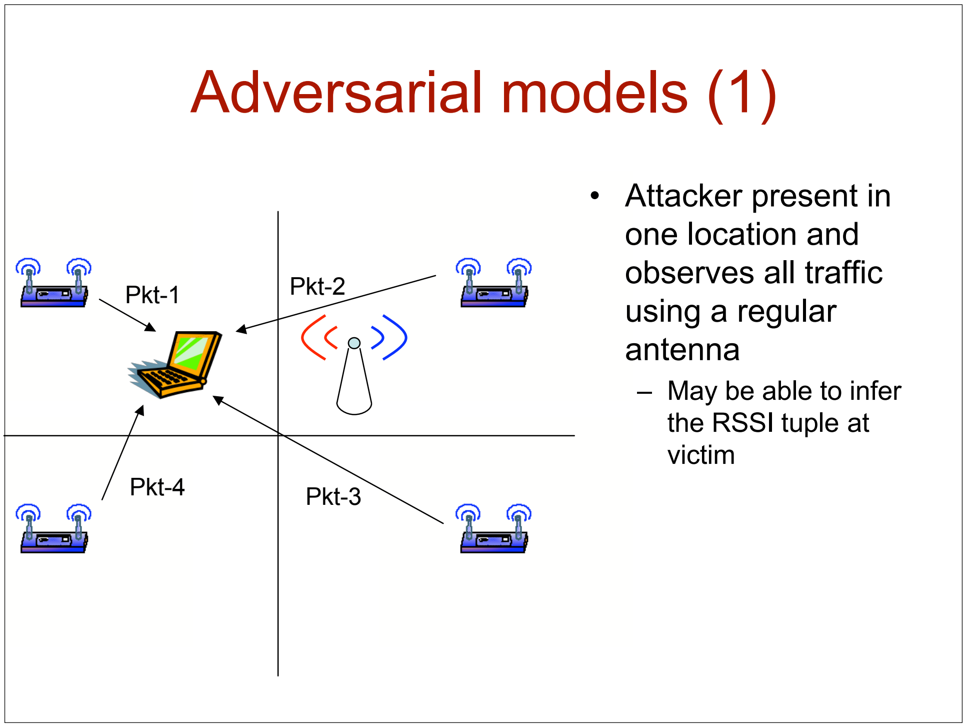
- Adversarial scenarios:
  - Integrity attacks:
    - Attacker pretends to be in a different location
    - Attacker makes the system believe that the victim is in a different location
  - Privacy attack:
    - Attacker infers location of victim and can track the victim

A specific localization approach



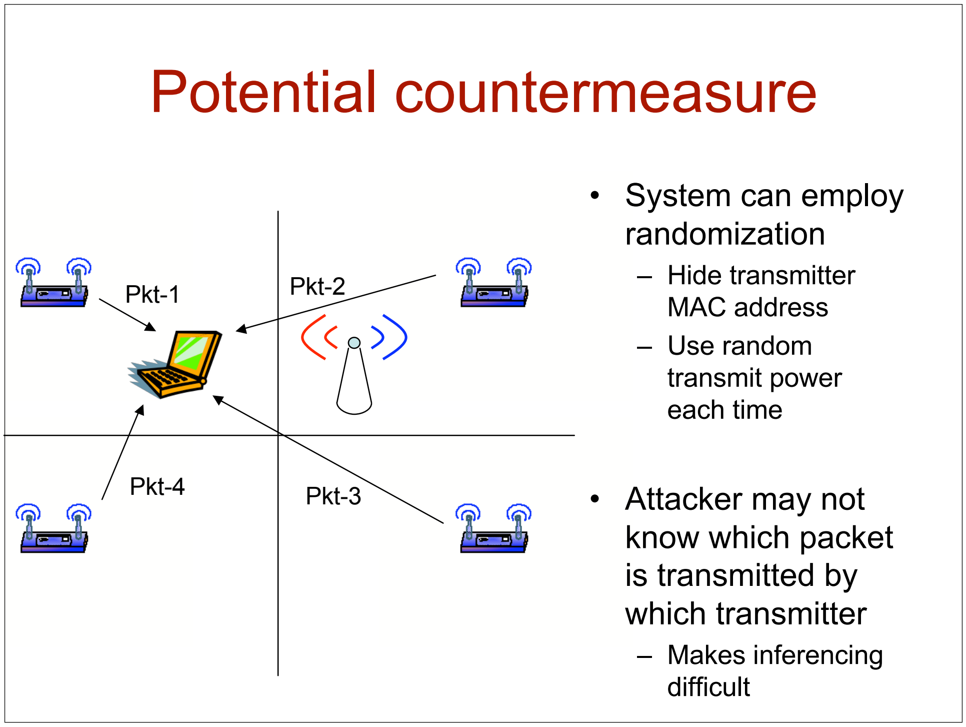
- Partition space into a grid
- System transmits some packets
- Participant reports RSSI tuple observed

- RSSI tuple is unique to a location and is the location signature



Adversarial models (1)

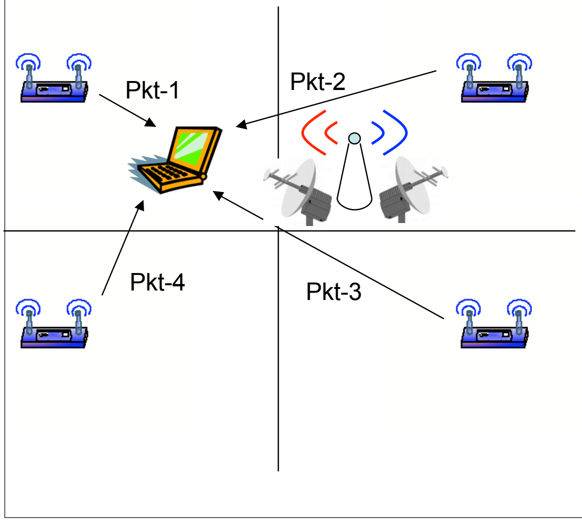
- Attacker present in one location and observes all traffic using a regular antenna
  - May be able to infer the RSSI tuple at victim



Potential countermeasure

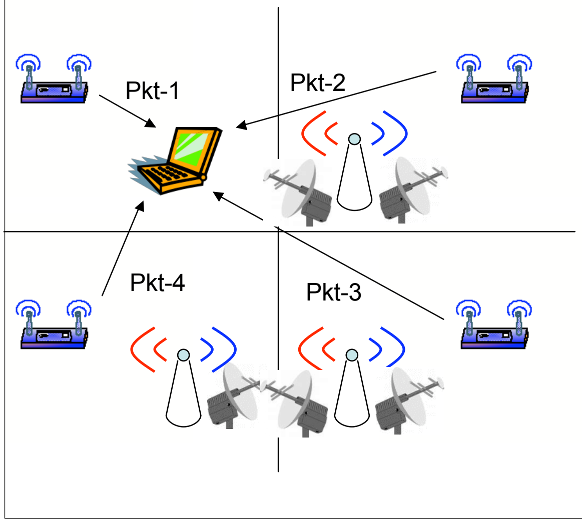
- System can employ randomization
  - Hide transmitter MAC address
  - Use random transmit power each time
- Attacker may not know which packet is transmitted by which transmitter
  - Makes inferencing difficult

Adversarial models (2)



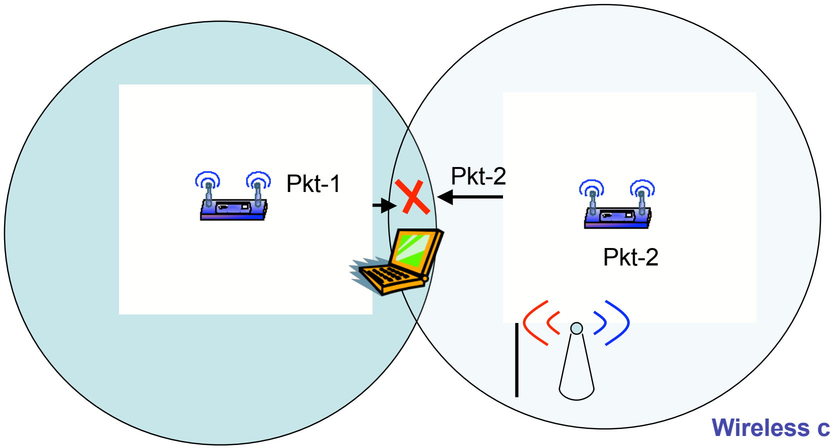
- Attacker able to tell Angle/Direction-of-Arrival
- Randomization may not help

Adversarial models (3)



- Even more sophisticated attacker
  - Present in multiple locations
  - Can allow attacker to have better location inference

More countermeasures



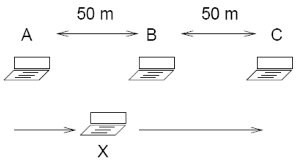
Wireless congruity
[HotMobile 2007]

Time-scheduled transmissions by the system that induce collisions may make inferencing harder

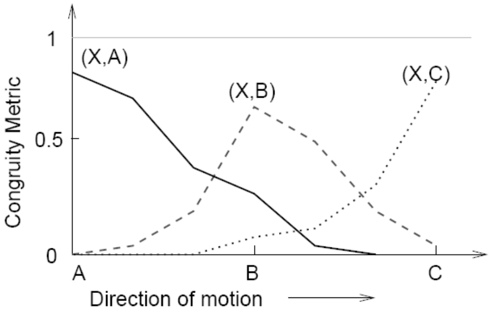
Wireless “congruity”

- Very robust in environments with high *entropy*
- First metric : $\zeta(A, B) = \dfrac{N_{AB}}{N_A + N_B - N_{AB}}$
- A is a trusted monitor, B is the user being authenticated

Congruity implies spatial vicinity



(a) Topology A



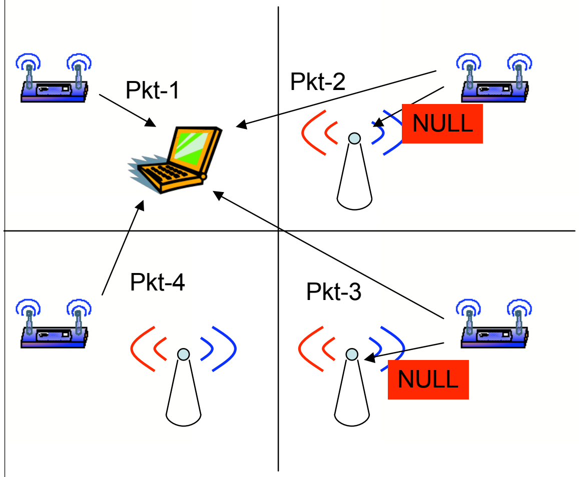
Based on the "congruity", it is possible to say

if X is near A, B or C

Optimizations

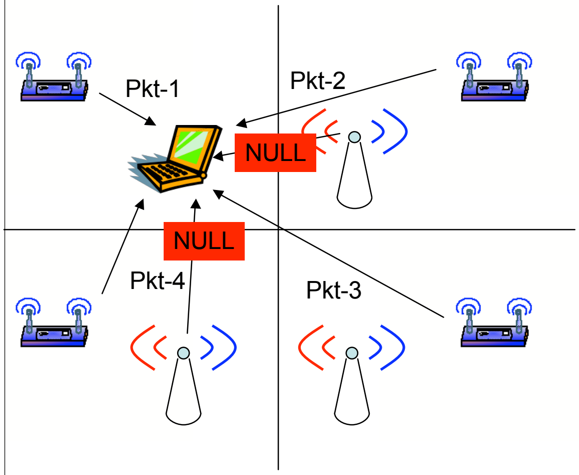
- Considering packets in *error* is useful
- Thresholding on RSSI of correctly received packets can also be useful
- Summary:
  - Wireless congruity is a promising approach to implement robust location authentication

More countermeasures



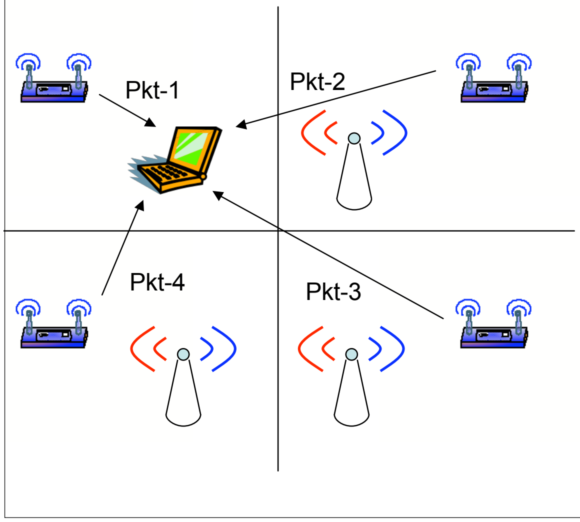
- Trusted system can use MIMO to create NULLs in certain directions
- Not always easy to determine directions to NULL
- Has other pitfalls

Adversarial models (4)



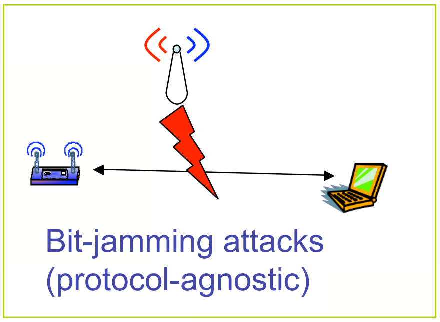
- Adversary can create NULLs at the victim as well

Adversarial models (5)

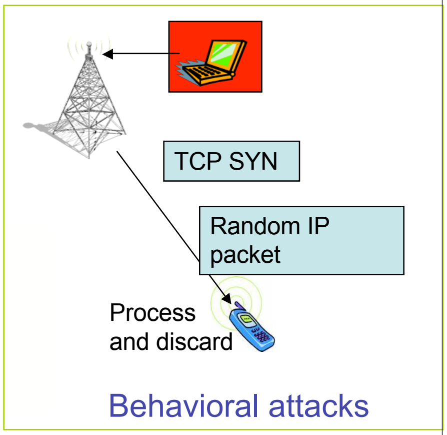


- Captured node in the system

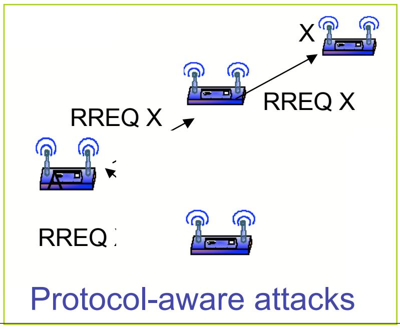
More adversarial scenarios



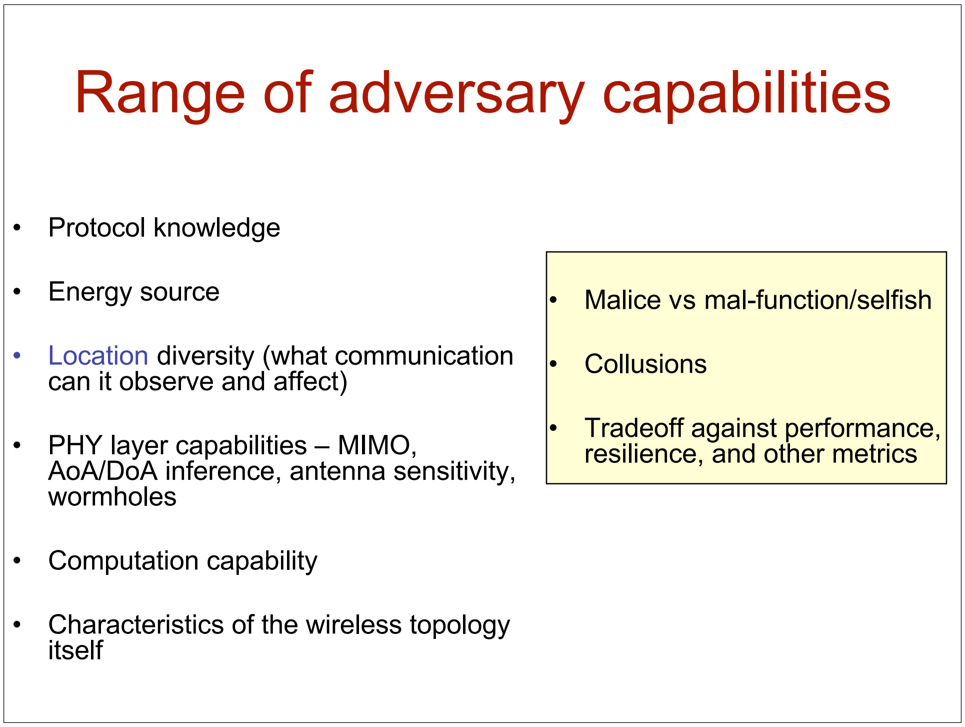
Bit-jamming attacks (protocol-agnostic)



Behavioral attacks



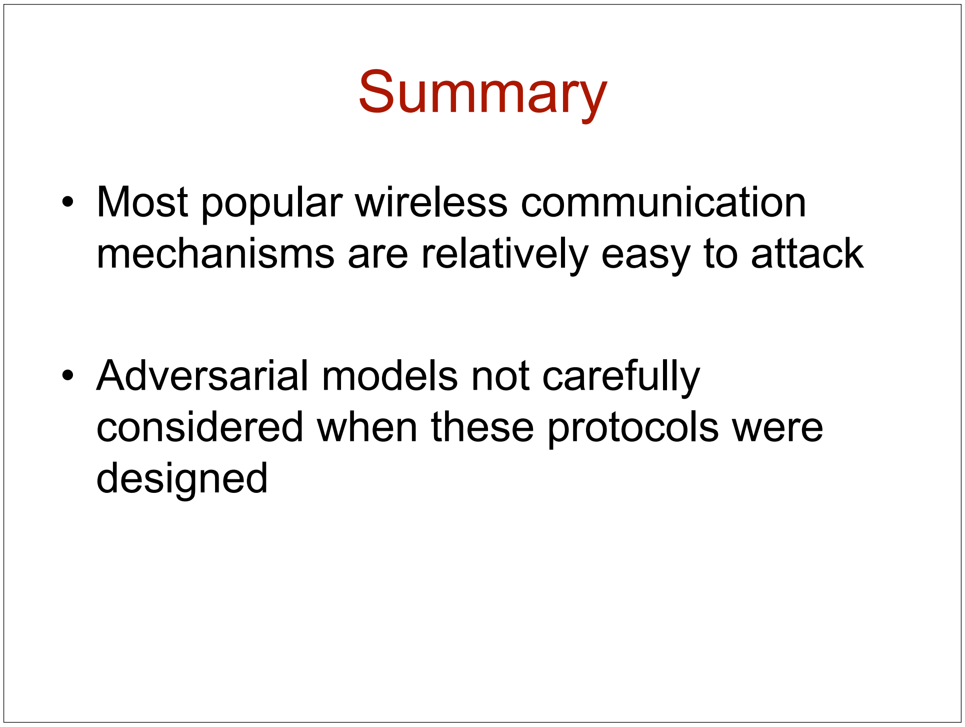
Protocol-aware attacks



Range of adversary capabilities

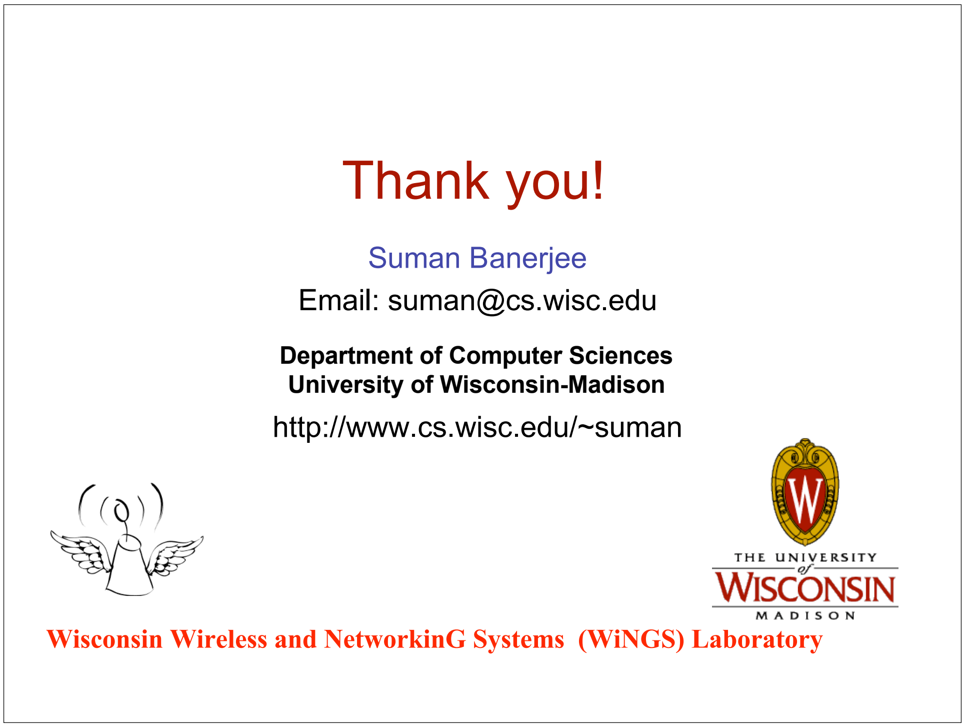
- Protocol knowledge
- Energy source
- Location diversity (what communication can it observe and affect)
- PHY layer capabilities – MIMO, AoA/DoA inference, antenna sensitivity, wormholes
- Computation capability
- Characteristics of the wireless topology itself

- Malice vs mal-function/selfish
- Collusions
- Tradeoff against performance, resilience, and other metrics



Summary

- Most popular wireless communication mechanisms are relatively easy to attack
- Adversarial models not carefully considered when these protocols were designed



Thank you!

Suman Banerjee

Email: suman@cs.wisc.edu

Department of Computer Sciences
University of Wisconsin-Madison

http://www.cs.wisc.edu/~suman

THE UNIVERSITY of WISCONSIN MADISON

Wisconsin Wireless and NetworkinG Systems (WiNGS) Laboratory

Session II – Languages and Software Engineering

Session Chair: Frank Mueller

Securing Embedded Software using Software Dynamic Translation

Position Paper for ARO Planning Workshop on
Embedded Systems and Network Security
February 22–23, 2007

Jack W. Davidson and Jason D. Hiser, University of Virginia, {jwd, jdh8d}@virginia.edu

1. Introduction

Embedded computer systems have become key building blocks of our nation's vital infrastructure. Critical systems controlled by embedded computer systems include communications systems, transportation and navigation systems, financial systems, medical systems, power distribution systems, and critical defense systems. Failure or compromise of such systems can have significant consequences including disruption of critical services, financial loss, and loss of life. Because critically functionality in embedded systems is increasingly implemented via software, three important research challenges for securing these systems is to provide protection from malicious observation, making them tamper resistant, and making them more resilient to unintentional and intentional memory errors in unsafe code that could be used to compromise an embedded system.

Unfortunately, securing embedded systems present several unique challenges not found in typical desktop or enterprise systems. Because of cost and power considerations, the execution environment for embedded software is often resource constrained—CPUs have limited processing power, there is often no memory management unit, and memory space is limited. Furthermore, embedded systems are frequently deployed in the field and must operate in physically insecure environments.

In this position paper, we discuss software dynamic translation and its potential for protecting software from malicious observation and tampering. While software dynamic translation can also be used to provide protection from unintentional and intentional memory errors that can be used to compromise an embedded system, even a brief discussion of the needed research and challenges in that area is beyond the scope of this paper.

2. Malicious Observation and Tampering

A trend in embedded systems is to provide functionality, which in the past was usually provided by hardware, via software. There are many advantages to using software instead of hardware to provide required functionality—reduced cost, increased flexibility, the ability to provide enhancements and patches, etc. However, moving functionality from hardware to software provides malicious parties easier access to valuable *intellectual property* (IP). In the context of this position paper, IP means information that an adversary could use for some malicious purpose (e.g., maliciously modifying a system, discovering a weakness that could be used to disable the system or render it ineffective, etc.) *Malicious observation* is the process of obtaining valuable IP. Of course, malicious observation could also be used to obtain valuable IP for commercial or financial advantage. Closely related to malicious observation is *malicious tampering*. Malicious tampering is the modification of software to change its intended behavior to achieve some malicious goal (e.g., cause damage, render the system ineffective, subvert some safeguards or licensing checks, etc.). Obviously, to intelligently tamper with a system, an attacker must have some knowledge about the operation of the system. Consequently malicious observation and tampering are closely related.

Because embedded systems are often deployed in hostile or insecure environments, one must assume that an attacker can gain physical access to the system. Consequently, an attacker can employ a variety of means to maliciously observe the operation of the software including the use of a virtual execution environment. The adversary can inspect, modify, or forge any information in the system. An adversary can run the program repeatedly and aggregate information from multiple runs of the program. In this extremely harsh environment, the adversary "holds all the cards" and with adequate time and resources, can gain a detailed understanding of the operation of the system.

3. Fundamental Limitations of Current Approaches

Current approaches to thwarting malicious observation have focused on making software hard to analyze statically. Addressing dynamic approaches to malicious observation has received little attention. While hardware approaches to preventing malicious observation and tampering can be effective in some contexts, hardware approaches may not be feasible within the cost- and resource-constraints imposed on embedded systems. In a similar vein, the few software approaches that have been proposed can require considerable computational resources and therefore are not applicable to embedded systems. Finally, much previous work has assumed an unrealistic threat model where an attacker does not have unfettered access to the system.

4. Software Dynamic Translation

A promising approach for addressing the very difficult problem of securing embedded software from malicious observation and tampering is to use software dynamic translation (SDT). SDT is a technology that enables software malleability and adaptivity at the binary instruction level by providing facilities for monitoring and dynamically modifying a program as it executes. SDT can affect an executing program by injecting new machine code, modifying existing code, or by monitoring and changing the control flow of the executing program. SDT has been successfully used in a variety of areas including binary translation, fast machine simulation, dynamic optimization, and to protect software from attacks that inject malicious code or attempt to change the normal execution flow of the program.

Using SDT, we envision a three-pronged approach to address this difficult challenge. First, SDT coupled with strong encryption technology can be used to make it difficult and costly for an adversary to statically and dynamically analyze embedded software. However, with physical access to the system and with adequate resources, a sophisticated and determined attacker could eventually obtain a detailed understanding of the software's operation. Therefore our second approach is to use SDT to make it difficult for an attacker to examine or modify a running system (including the one that an attacker might have obtained for malicious observation). Third, SDT is used to create diverse versions of the software to make it difficult to aggregate information across different executions of the system. Dynamic diversity also ensures that knowledge gained by capturing and observing one instance of a system is not applicable to any other deployed instance. Thus, even if an attacker can determine what modifications to make to achieve their goal for one instance of the system (i.e., the system to which they have physical access), this knowledge is not useful for attacking other instances of that system.

5. Milestones

For SDT to be applicable to embedded systems, it must be demonstrated that SDT can be applied to embedded software running on typical embedded processors. Preliminary results for the ARM processor using some widely used embedded benchmarks indicate that SDT can be efficiently accomplished on an embedded system. Research adapting SDT to other architectures and other types of systems (e.g., hard real-time systems, reactive systems, etc.) needs to be carried out. Of paramount importance is the ability to perform SDT on resource-constrained systems without excessive overhead.

A difficult problem is the assessment of the effectiveness of techniques to provide protection against malicious observation and tampering. Of particular concern is that there are no objective metrics or models for assessing the effectiveness of techniques to protect software against malicious observation when an intelligent human adversary is guiding the effort (which will almost always be the case). Development of metrics and models for assessing the effectiveness of techniques used to protect embedded software against malicious observation and tampering is critical.

While hardware solutions to the malicious observation and tampering problem have been proposed, they can greatly increase the cost of a system. For systems where deployment means the delivery of thousands of devices, the cost of a comprehensive hardware solution may not be feasible. However, modest hardware additions designed to support SDT-based solutions to malicious observation and tampering may be cost effective and provide a higher level of protection than simple hardware- or SDT-based protection mechanisms alone.

Position Paper: Deeply Embedded Survivability

Philip Koopman, Jennifer Black, Theresa Maxino
Carnegie Mellon University
{koopman, jenm, maxino}@cmu.edu

Abstract

This position paper identifies three significant research challenges in support of deeply embedded system survivability: achieving dependability at the enterprise/embedded interface gateway, finding a viable security patch approach for embedded systems, and surviving run-time software faults.

1. Introduction

Deeply embedded systems consist of one or more embedded systems connected to an enterprise system or to the Internet (e.g., [3]). To be survivable, such systems must continue to function in the face of faults, whether accidental or malicious, and whether the faults are caused by design errors or unexpected operating conditions. Embedded system survivability can be more challenging than enterprise survivability because embedded systems may not be able to perform frequent reboots, incorporate weekly patches, transfer large amounts of data, or be cared for by trained system administrators. Beyond this, the different natures of embedded control vs. enterprise systems present fundamental limitations to applying known techniques from either area to the other. [1]

2. Fundamental limitations

2.1 Time triggered to event triggered interfaces

A fundamental limitation to achieving deeply embedded system survivability is the inherent mismatch between time triggered and event triggered systems.

Embedded systems are often "time triggered," meaning that they perform periodic computations and messaging in support of hard deadlines (e.g., [2]). Because of the dramatically different needs of real time control systems compared to desktop computing, they often use specialized network protocols such as CAN that provide low-cost, but low-bandwidth solutions optimized for very short messages (often 100 bits or fewer per message with network speeds on the order of 1 Mbit/sec).

Enterprise systems, in contrast, are usually characterized as "event triggered" systems with much larger, sporadic events, and typically have orders of magnitude more CPU power and network bandwidth.

The interface between the embedded and enterprise sides of a deeply embedded system is usually in the form of a "gateway" that provides a bidirectional transition between the time triggered and event triggered worlds. Given sufficient resources, each computing paradigm can be made to simulate the other. Event triggered systems can schedule events periodically to simulate time triggered operation. Time triggered systems can schedule periods so fast that they don't miss events. But, those approaches only work in the fault-free case.

Deeply embedded system gateways will encounter fundamental limitations when attempting to map faults and responses in one computing paradigm into the other computing paradigm. For example, what happens when event triggered messages are clumped in transit, and arrive faster than the minimum inter-arrival rate assumed by the time triggered side of the gateway? Queues in the gateway provide only a partial solution, and can cause problems when the system encounters queue overflow or system instability as a result of queue lag time.

In the other direction, time triggered messages that contain too much value jitter can defeat whatever low pass filters are in place at the gateway and can potentially flood the enterprise system with messages. Leaky buckets and other throttling methods can provide some relief, but are not necessarily able to do the right thing in those cases where an event shower is representative of a true emergency situation rather than a fault or attack.

Despite a lack of understanding of these fundamental issues, deeply embedded system gateways are already being deployed, sometimes in critical systems.

2.2 Limits to the patch mentality

The approach of using security patches to address emergent attacks is pervasive in the desktop computing environment. Embedded systems have fundamentally different constraints that make patching difficult.

Safety critical systems must be recertified each time critical software is updated. Doing so is usually a costly and time-consuming process. Quick-turnaround security patches are currently impracticable if they affect critical code. Unfortunately, many embedded systems are designed in such a way that all their code is effectively critical (i.e., any change to the code might affect critical properties, so it must all be assumed to be critical).

Strategies to isolate critical from non-critical software on the same CPU are still a subject of research.

An additional issue with patching embedded systems is that many of them have a zero down-time requirement. Maintenance reboots and physical operator intervention are simply unacceptable in many unattended applications.

Finally, patching approaches typically assume that the owner of a system is trustworthy. This is often not the case in embedded systems. For example, it is relatively common for sports car owners to install engine controller software that circumvents pollution emission and fuel economy controls as a way to get more performance.

2.3 Limits to the perfect software mentality

Much research in computer science is based on the laudable goal of creating perfect software. Industry practices also employ the assumption that "perfection" (or a close approximation thereof) can be achieved by identifying all the "important" bugs and removing them.

In the real world, very few application domains have the time and resources to deploy low defect rate software. Getting the highest software quality possible within time and budget is certainly important. But, spending exponentially increasing resources to chase down the last few bugs is usually impractical. Instead, it might make more sense to spend a small fraction of available resources providing ways to survive bugs that will inevitably be encountered, rather than throwing all resources at an attempt to achieve absolute perfection.

3. Research challenges

There are several research challenges that stem from the limitations just discussed. They are:

Understand what goes into the embedded/ enterprise gateway. While some combination of queues and message filters can work in the fault-free case, mapping fault manifestations and survivability mechanisms across the time triggered to event triggered interface provides fundamental research challenges.

Make patching of critical embedded software viable. Patching of unattended, critical embedded systems provides fundamental challenges that aren't encountered in most desktop systems. Creating patching approaches that maintain system integrity promises to be difficult.

Increase system survivability by tolerating inevitable software defects. Software defects are inevitable in most fielded systems. In some cases these defects will result in security vulnerabilities. In others they will result in failures to maintain critical system properties. Making software faults more survivable could offer improved cost effectiveness and reduced system fragility.

4. Promising innovations and abstractions

4.1 Safety invariants

Safety invariants, which are formal expressions of critical system properties that must hold true, offer new promise for increasing system survivability. Traditionally, analysis and testing are used to ensure the invariants are never violated. But, these techniques only work for the systems that are modeled (which are usually fault-free systems). One could also check safety invariants at run time to detect when a fault has occurred that is severe enough to compromise system safety. Safety invariant checks could act as failure detectors that activate recovery or safe shutdown mechanisms.

4.2 Graceful degradation

The term graceful degradation encompasses several meanings. The term was coined to describe modular redundancy in fault tolerant computing, and later evolved to encompass failover strategies and functional diversity. More recently, the term has been used to describe performability tradeoffs in Quality of Service research. The notion of providing systems that can partially work rather than only be fully working or fully failed is essential to achieving cost-effective survivability.

5. Possible Milestones

Survivability is an emerging research area, with the current emphasis more on understanding fundamental problems rather than on comprehensive solutions. Long-term milestones should include discovering fundamental tradeoffs, impossibility results, and workarounds applicable to realistic systems. Short term research milestones should emphasize characterizing practical limitations and exploring techniques to offer near-term improvement to system builders.

6. Acknowledgements

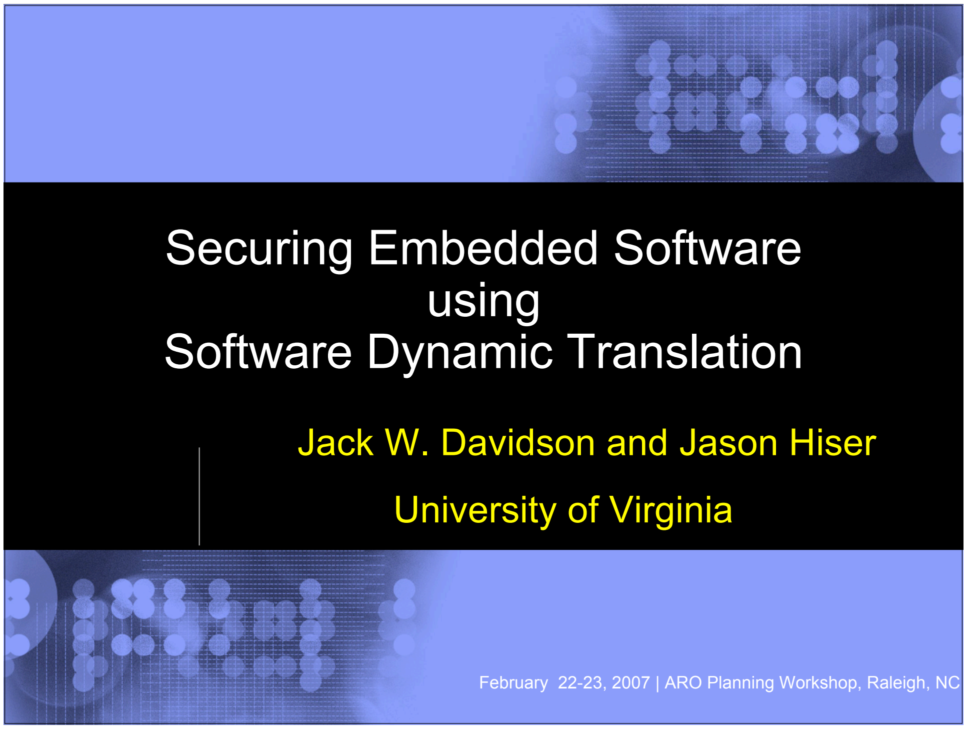
This work had been funded by the General Motors Collaborative Research Laboratory at Carnegie Mellon University and the Pennsylvania Infrastructure Technology Alliance.

7. References

[1] Koopman, P., Morris, J. & Narasimhan, P., "Challenges in Deeply Networked System Survivability," *NATO Advanced Research Workshop On Security and Embedded Systems*, August 2005

[2] Kopetz, H., *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer, 1997.

[3] Tennenhouse, D., "Proactive Computing," *Comm. ACM*, 43(5): 43-50, May 2000.

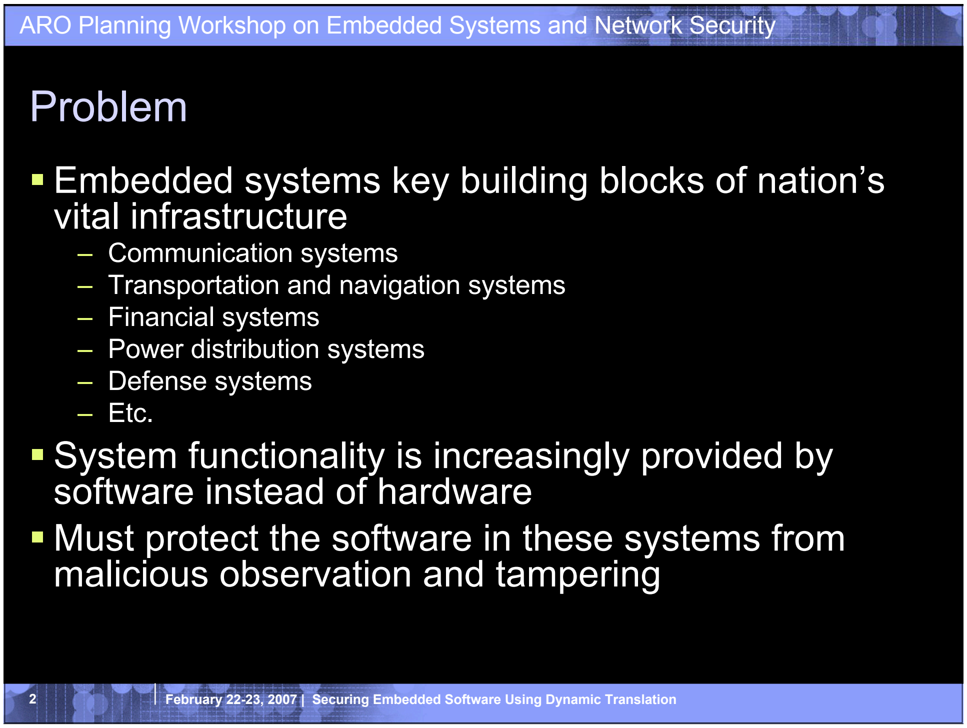


Securing Embedded Software using Software Dynamic Translation

Jack W. Davidson and Jason Hiser

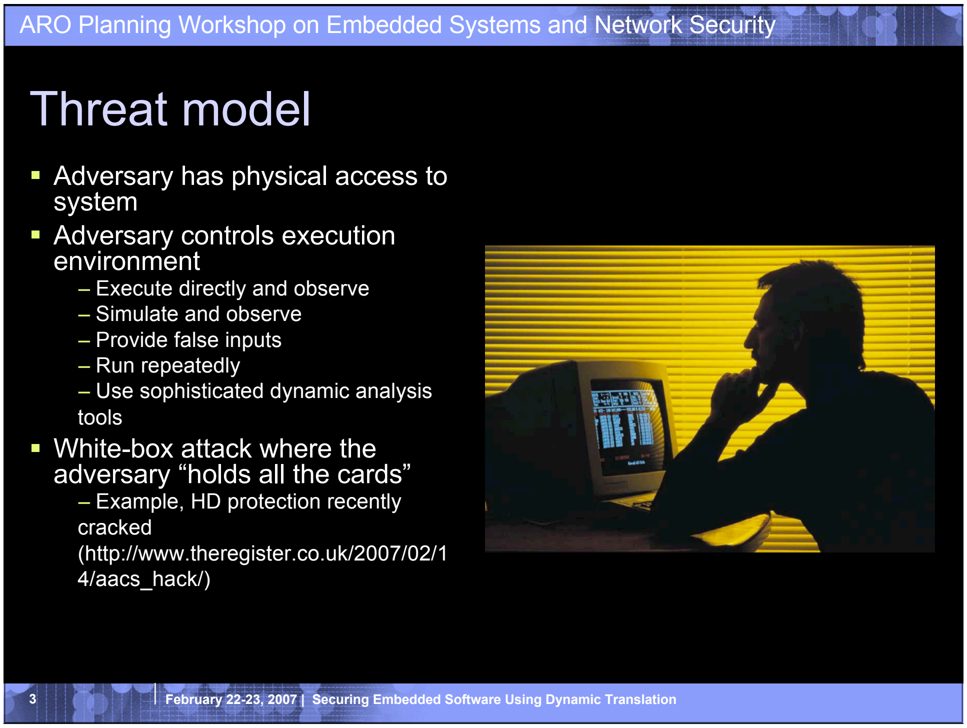
University of Virginia

February 22-23, 2007 | ARO Planning Workshop, Raleigh, NC



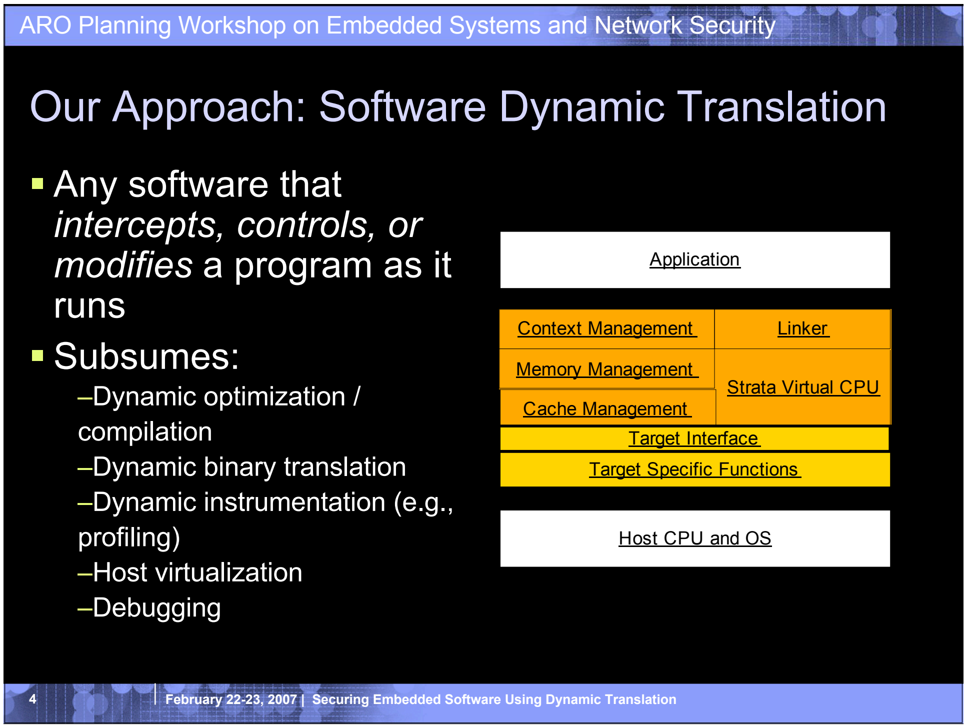
Problem

- Embedded systems key building blocks of nation’s vital infrastructure
  - Communication systems
  - Transportation and navigation systems
  - Financial systems
  - Power distribution systems
  - Defense systems
  - Etc.
- System functionality is increasingly provided by software instead of hardware
- Must protect the software in these systems from malicious observation and tampering



Threat model

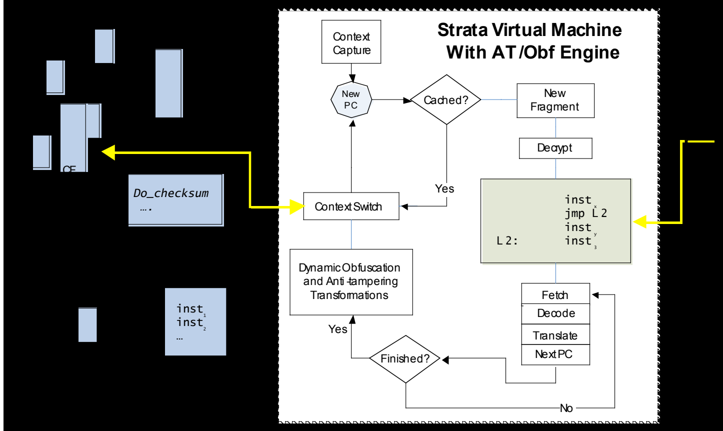
- Adversary has physical access to system
- Adversary controls execution environment
  - – Execute directly and observe
  - – Simulate and observe
  - – Provide false inputs
  - – Run repeatedly
  - – Use sophisticated dynamic analysis tools
- White-box attack where the adversary "holds all the cards"
  - – Example, HD protection recently cracked (http://www.theregister.co.uk/2007/02/14/aacs_hack/)



Our Approach: Software Dynamic Translation

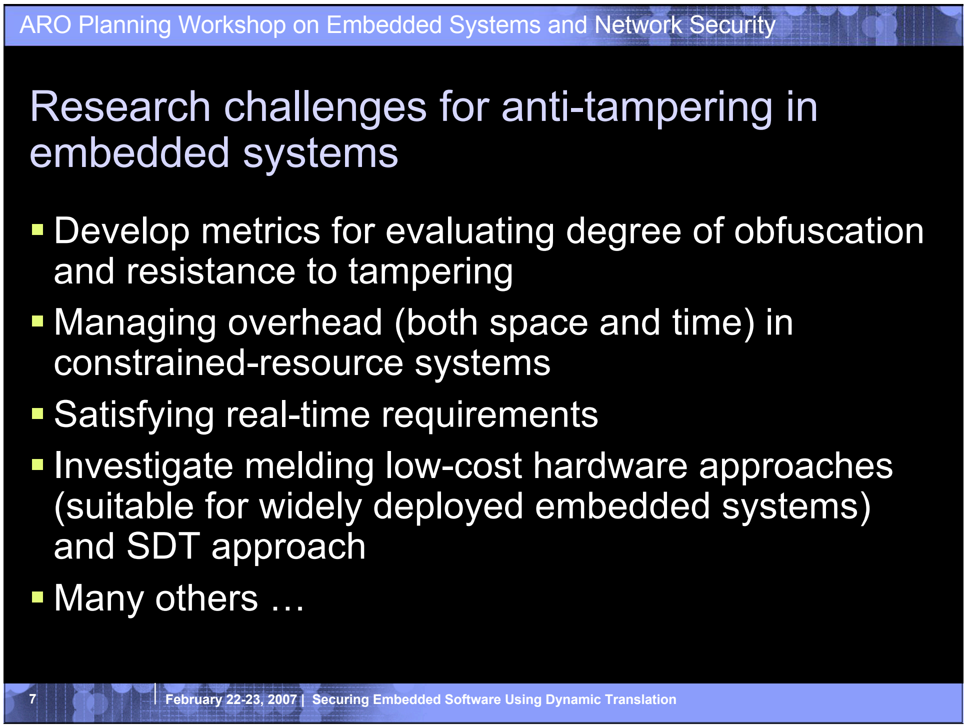
- Any software that *intercepts, controls, or modifies* a program as it runs
- Subsumes:
  - –Dynamic optimization / compilation
  - –Dynamic binary translation
  - –Dynamic instrumentation (e.g., profiling)
  - –Host virtualization
  - –Debugging

Using SDT for Obfuscation and Anti-tampering



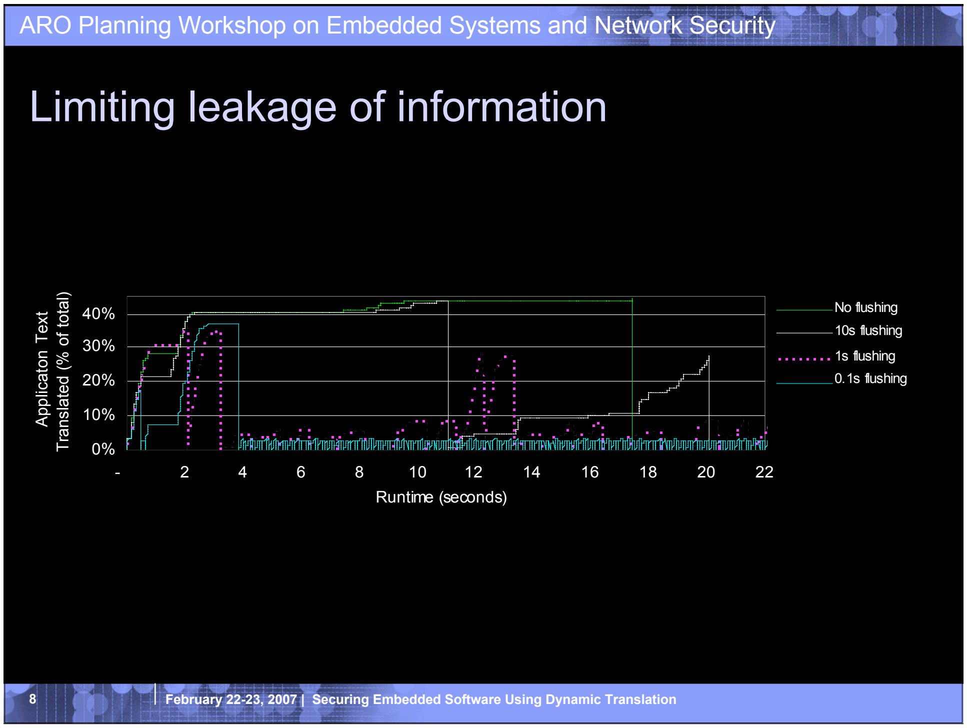
Benefits

- Prevents static disassembly and analysis
  - Code is encrypted on disk
  - Must run SDT system to materialize code
- Provides dynamic obfuscation of code
  - Natural obfuscation of code by SDT system
  - Dynamically apply obfuscations
- Prevents manipulation of running code
  - Guards prevent changing application or SDT system
  - Fragment cache is protected
- Limits leakage of information
  - Flush fragment cache frequently
  - Multiple runs provide less advantage to attacker
- Provides diverse implementations
  - Dynamic transformations applied randomly
  - Weakness or vulnerability discovered in one instance not necessarily exploitable in other instances

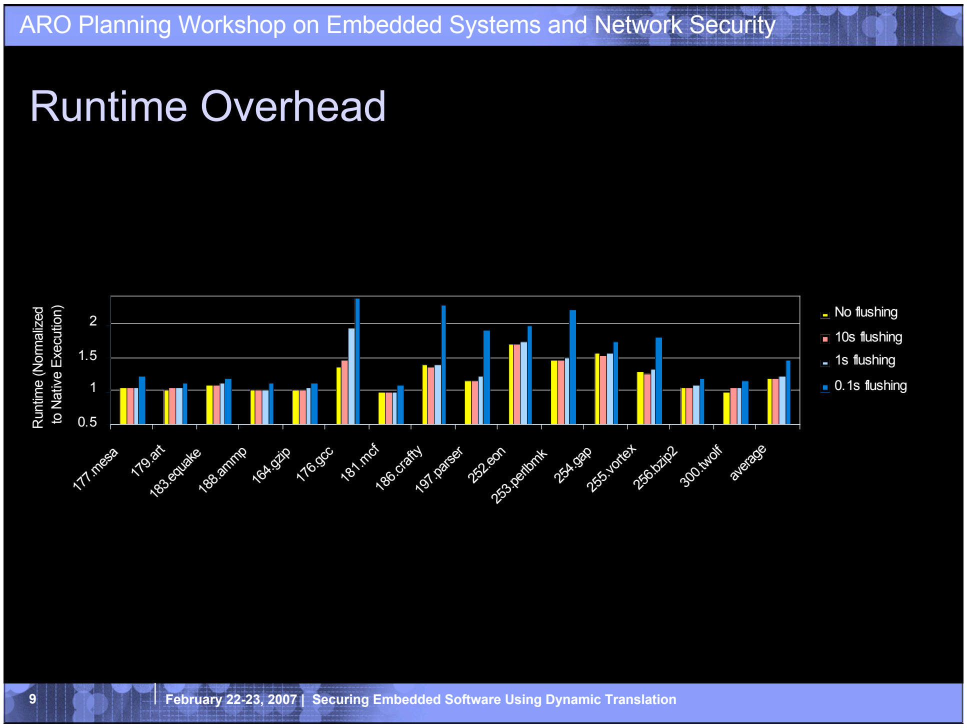


Research challenges for anti-tampering in embedded systems

- Develop metrics for evaluating degree of obfuscation and resistance to tampering
- Managing overhead (both space and time) in constrained-resource systems
- Satisfying real-time requirements
- Investigate melding low-cost hardware approaches (suitable for widely deployed embedded systems) and SDT approach
- Many others …



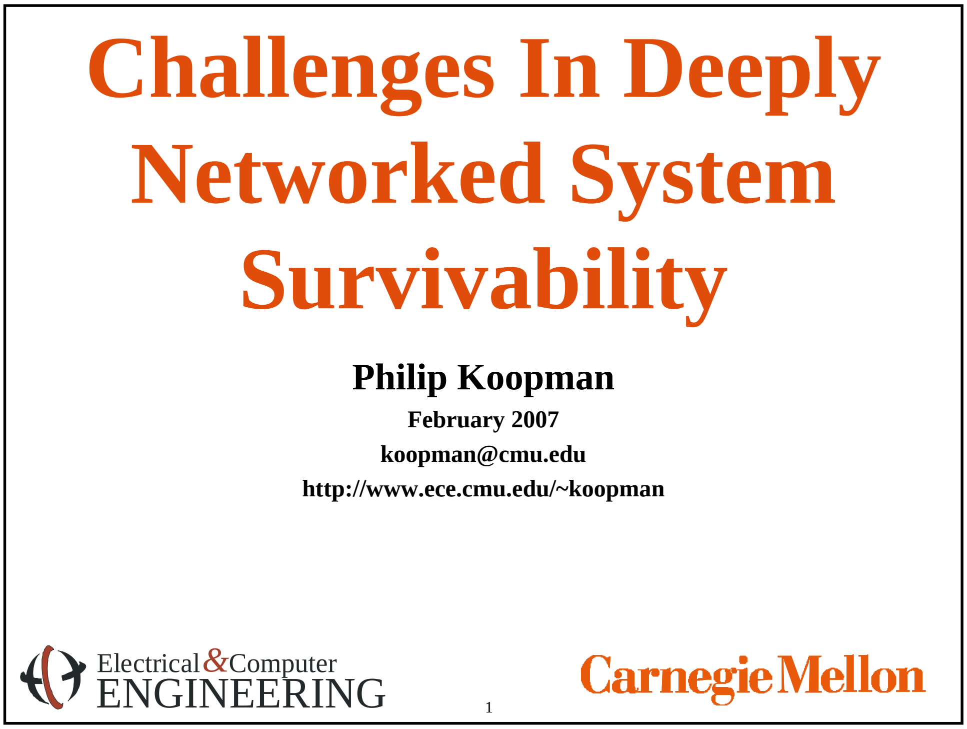
Limiting leakage of information



ARO Planning Workshop on Embedded Systems and Network Security

Runtime Overhead

9 | February 22-23, 2007 | Securing Embedded Software Using Dynamic Translation



Challenges In Deeply Networked System Survivability

Philip Koopman

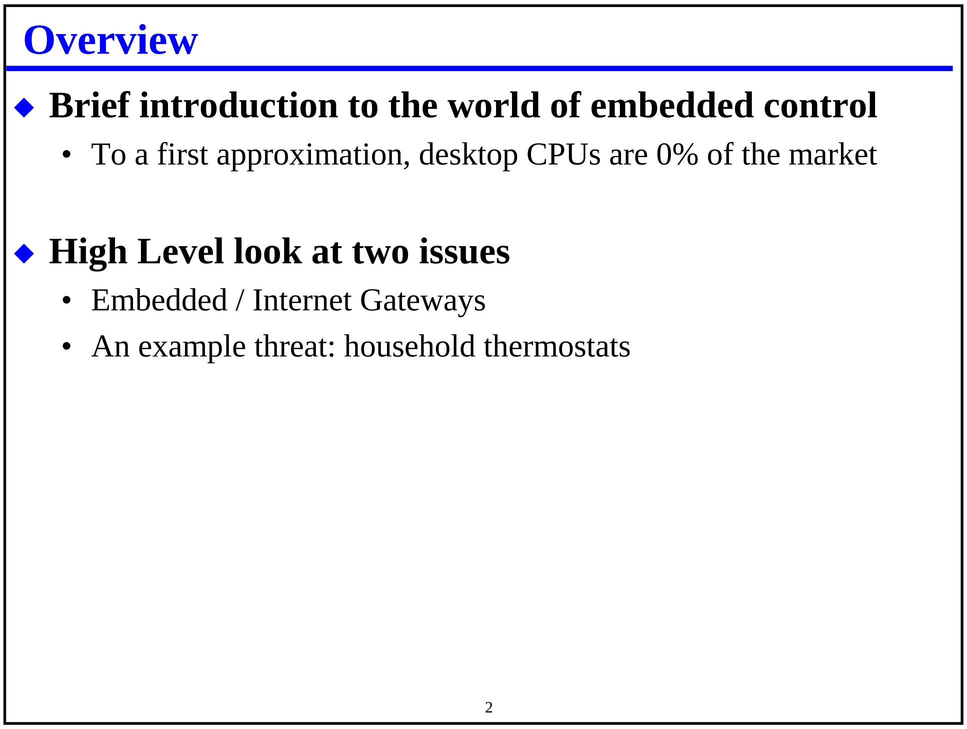
February 2007

koopman@cmu.edu

http://www.ece.cmu.edu/~koopman

Electrical & Computer ENGINEERING

Carnegie Mellon

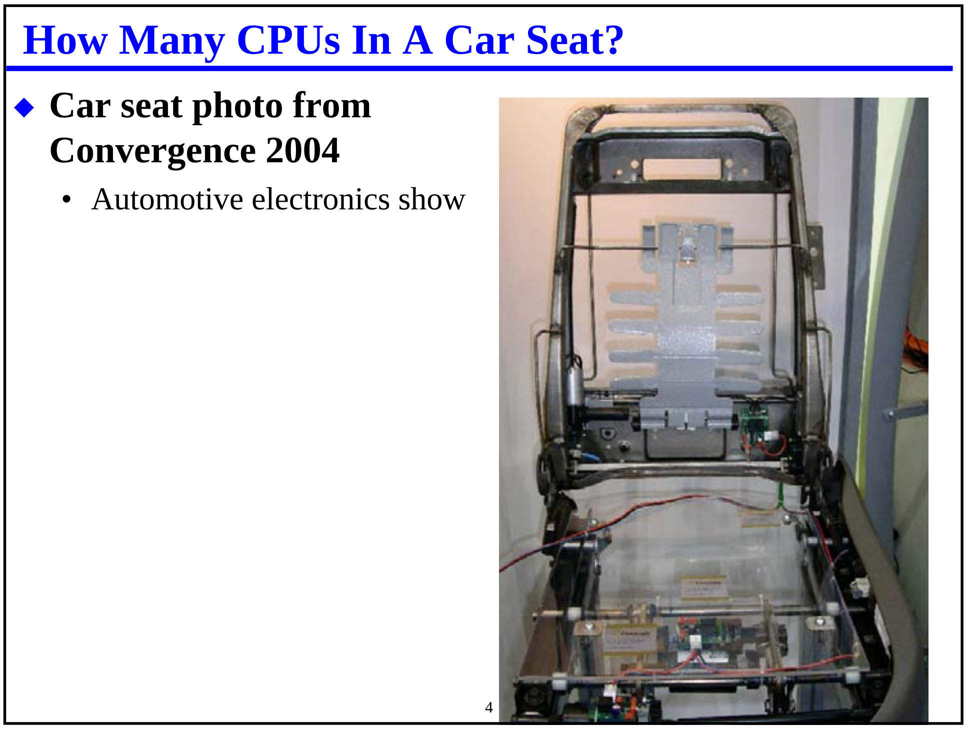


Overview

- **Brief introduction to the world of embedded control**
  - To a first approximation, desktop CPUs are 0% of the market
- **High Level look at two issues**
  - Embedded / Internet Gateways
  - An example threat: household thermostats

My Experience in Embedded Systems

3



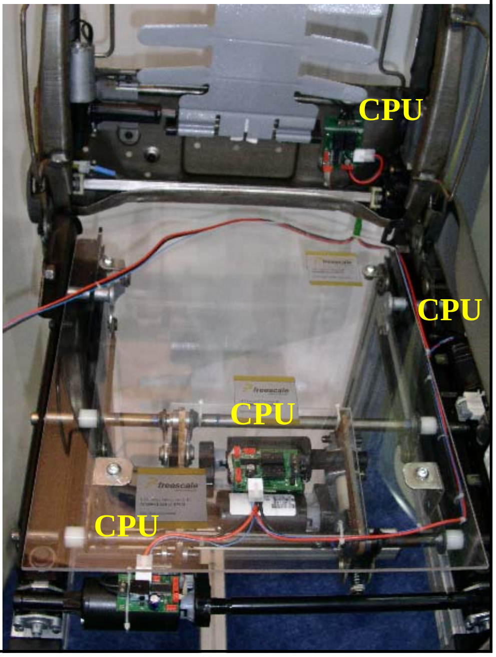
How Many CPUs In A Car Seat?

- **Car seat photo from Convergence 2004**
  - Automotive electronics show

4

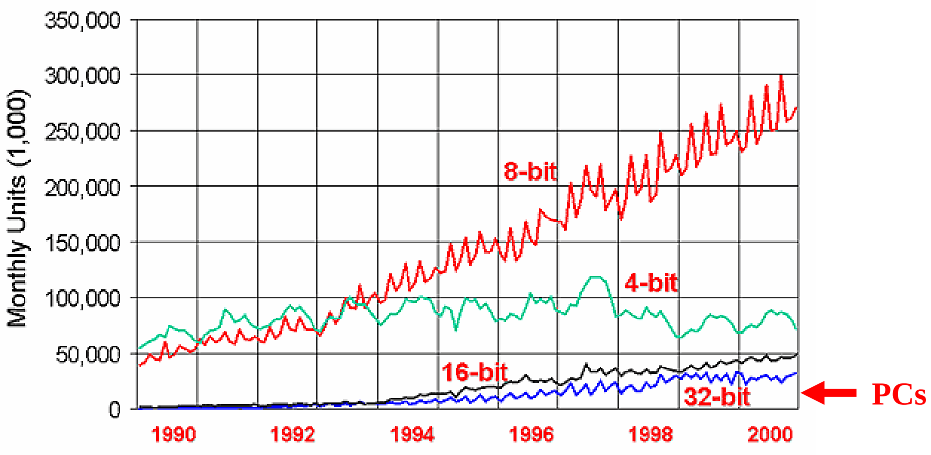
Car Seat Network (no kidding)

- Low speed LIN network to connect seat motion control nodes
- This is a distributed embedded system!
  - Front-back motion
  - Seat tilt motion
  - Lumbar support
  - Control button interface
  - Connects to body controls network beyond seat for per-driver customization



Microprocessor Unit Sales

All types, all markets worldwide



Source: WSTS

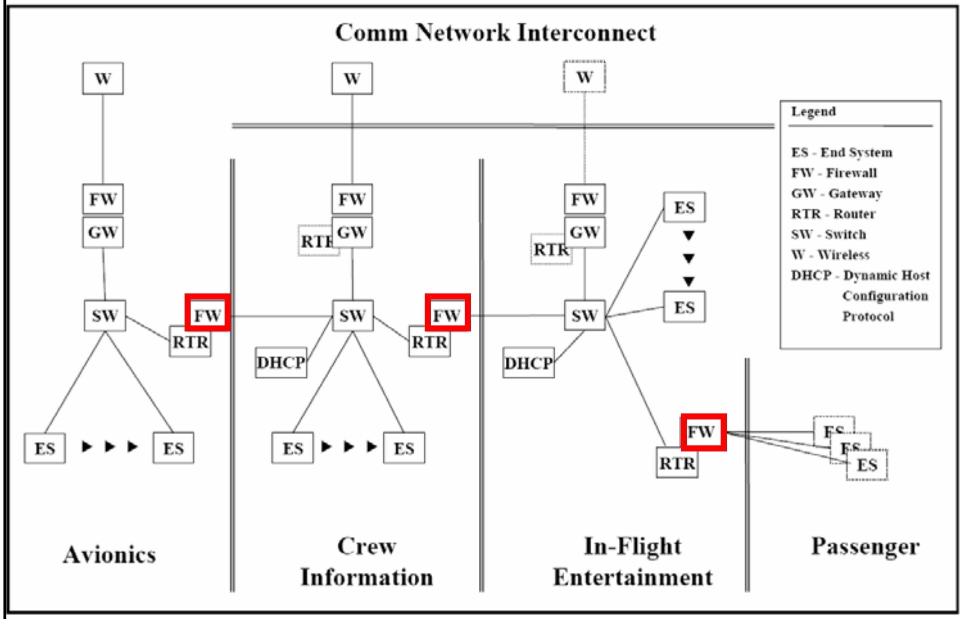
15 Million PCs per month in 2004 (15,000 on this graph)

Trend: External Connectivity

- **Safety critical subsystems will be connected to external networks (directly or indirectly)**
  - German proposal:
    wireless networks control car's max. speed
  - E-enabled aircraft architecture (next slide)

Computer graphics by I3M

[Airbus 2004] A-380 scheduled to enter service in 2006



Legend

ES - End System
FW - Firewall
GW - Gateway
RTR - Router
SW - Switch
W - Wireless
DHCP - Dynamic Host Configuration Protocol

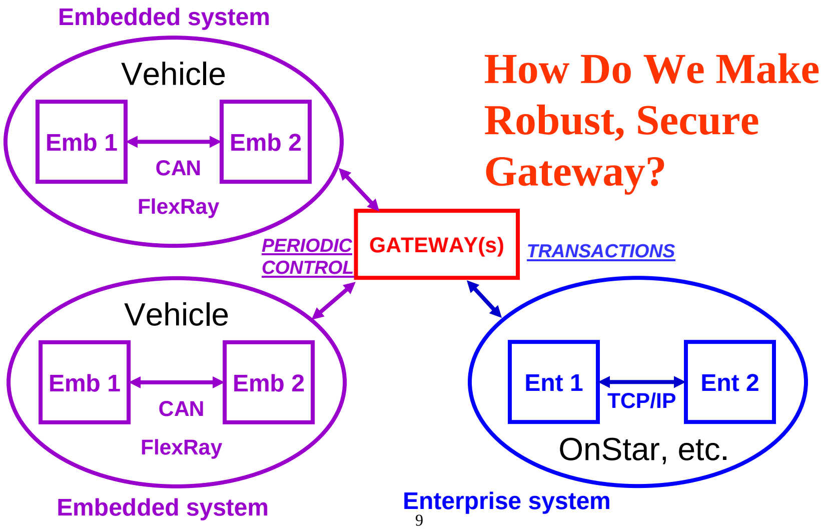
Wargo & Chas, 2003, proposed Airbus A-380 architecture

Passenger laptops are 3 Firewalls away from flight controls!

Deeply Embedded System Gateway

Enterprise system + Embedded System =
"Deeply Embedded System"

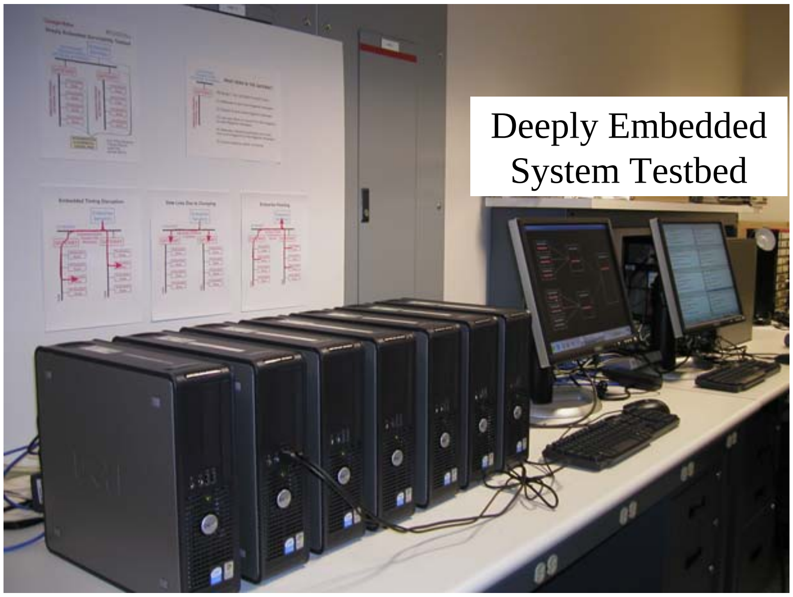
How Do We Make A Robust, Secure Gateway?



Research Area: Embedded/Internet Gateway

- **What happens at the embedded/internet interface?**
  - Fault propagation across the gateway presents fundamental challenges

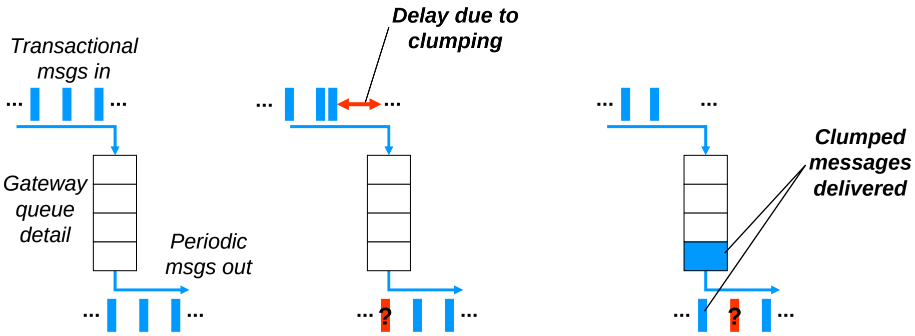
Embedded Side	GATEWAY	Enterprise Side
Control-oriented		Transaction-oriented
Time Triggered		Event Triggered
Continuous		Discrete
Real Time		Mostly not Real Time
Periodic Messages		Aperiodic Messages
Short Messages		Longer messages
Roll-forward		Rollback
Lower cost		Higher cost



Deeply Embedded System Testbed

Initial Experiment: Queue overflow

- How having a long queue can cause you to operate on stale data



Ideal case: Queue is empty in the steady state

Clumping delay leads to missed deadline for periodic messages

Need a policy for dealing with having no message to send.

Two messages delivered, so a message is stuck in the queue.

Now all the messages delivered to the embedded system are 1 period old.

Deeply Embedded Scary Scenario

- **Consider the lowly thermostat**
  - Koopman, P., "Embedded System Security," *IEEE Computer*, July 2004.
- **Trends:**
  - Internet-enabled
  - Connection to utility companies for grid load management
- **Proliphix makes an Internet Thermostat**
  - (But it we're not saying that system has these vulnerabilities!)
  - Somebody else makes one almost exactly like this, deployed July 2005

Waste Energy Attack

- **"I'm coming home" function**
  - Ability to tell thermostat to warm up/cool down house if you come home early from work, or return from a trip
  - Save energy when you're gone; have a comfy house when you return
  - Implement via web interface or SMS gateway
- **Attack: send a false "coming home" message**
  - Causes increase in utility bill for house owner
  - If a widespread attack, causes increased US energy usage/cause grid failure
  - Easily countered(?) – if designers think to do it!
    - Note that playback attack is possible – more than just encryption of an unchanging message is required!

Discomfort Attack

- **Remotely activated energy saver function**
  - Remotely activated energy reduction to avoid grid overload
  - Tell house "I'll be home late"
  - Saves energy / prevents grid overload when house empty

- **Attack: send a false "energy saver" command**
  - Will designers think of this one?
  - Some utilities broadcast energy saver commands via radio
    - In some cases, air conditioning is completely disabled
    - Is it secure??
  - Consequences higher for individual than for waste energy attack
    - Possibly broken pipes from freezing in winter
    - Possibly injured/dead pets from overheating in summer

Energy Auction Scenario

- **What if power company optimizes energy use?**
  - Slightly adjust duty cycles to smooth load (pre-cool/pre-heat in anticipation of hotest/coldest daily temperatures)
  - Offer everyone the chance to save money if they volunteer for slight cutbacks during peak times of day
  - Avoid brownouts by implementing heat/cool duty cycle limits for everyone

- **You could even do real time energy auctions**
  - Set thermostat by "dollars per day" instead of by temperature
    - More dollars gives more comfort
  - Power company adjusts energy cost continuously throughout day
  - Thermostats manage house as a thermal reservoir

Energy Auction Attacks – Naïve Version

- **What if someone broke into all the thermostats?**
  - Set dollar per day value to maximum, ignoring user settings
    - Surprise! Next utility bill will be unpleasant
  - Turn on all thermostats to maximum
    - Could overload power grid
  - Pulse all thermostats in a synchronized way
    - Could synchronized transients destabilize the power grid?

Energy Auction Attacks – Scary Version

- **What if someone broke into the energy auction server?**
  - If you set energy cost to nearly-free, everyone turns on at once to grab the cheap power
  - Guess what – enterprise computer could have indirect control of thousands of embedded systems!
  - Someday soon, almost “everything” will be “embedded,” at least indirectly

“Unique” Embedded System Requirements

Embedded systems:

- **Are actually supposed to work**
  - Do you want to perform a workaround for your water heater?
  - Often have 24x7 requirements – zero down time
- **Often are safety critical**
  - Have you ever ridden in a fully automated train/peoplemover? (or an elevator?)
- **Are very cost sensitive & resource constrained**
  - A $0.50 CPU can’t run a “big” OS with full security features
- **Don’t have a sysadmin**
  - Who’s the sysadmin for your DVD player?
  - The owner is often negligent, or even a malicious attacker

Session III – Software Security

Session Chair: Purush Iyer

Software Security Issues in Embedded Systems

Somesh Jha
Computer Sciences Department,
University of Wisconsin, Madison, WI 53706.

1 Introduction

Embedded systems and networks are becoming increasingly prevalent in critical sectors, such as medical and defense sectors. Therefore, malicious or accidental failures in embedded systems can have dire consequences. Hence, the integrity of embedded software infrastructures, such as configuration and code, is of paramount importance. The autonomous nature of embedded systems also poses new challenges in the context of system integrity. Since embedded systems are reactive, unexpected or malicious environment events or can cause failures, which can have dire consequences in critical sectors. Embedded systems and networks also often have to operate autonomously in a dynamic environment. Therefore, an embedded system has to adapt its behavior to the change in environment or the overall goal. Unauthorized or unverified updates to the infrastructure of an embedded system can also compromise its integrity.

In recent years, there have been significant advances in the area of software security. There have been various techniques developed in the context of software security, such as automated signature generation, vulnerability assessment, and detecting malicious behavior. However, all these techniques are not directly applicable in the context of embedded systems because of following reasons:

- *Dynamic and configurable environment:* Embedded systems are generally deployed in environments that are highly dynamic and configurable. For example, the environment of an embedded system deployed in a battlefield is extremely dynamic.

- *Changing functional requirements:* Functional requirements of an embedded system change over time. Functional requirements of an embedded system deployed in a battleship change with mission of the operation.

- *Interconnected network of components:* Frequently an embedded system is a complex network of components. Therefore, a malicious or accidental fault in a component can lead to a complex cascade of events in the network.

- *Recovery is paramount:* Generally, techniques developed in the realm of software security focus on detection and prevention. Embedded systems are frequently deployed in mission critical applications where consequences of failures can be dire. Therefore, recovery from failures is extremely important in the context of embedded systems.

2 Promising Research Directions

Extending existing techniques in software security to handle the four abovementioned characteristics of embedded systems is an important research direction. I will provide details of two such research directions.

- *Vulnerability assessment and prevention in presence of a dynamic environment:* Existing techniques for vulnerability assessment have been developed for systems (such as servers) whose environments are relatively static. Extending dynamic and static analysis techniques for vulnerability assessment and prevention for systems with dynamic environments is a very interesting research direction. I envision that existing techniques will have to be extended to incorporate specification of the environment. In this context, an interesting research direction would be to generate vulnerability signatures [2, 6] for systems with dynamic environments. I envision the signatures in this case will be parametrized by a specification of the environment, i.e., signatures will only be valid if certain environment conditions are satisfied.

- *Recovery from malicious or accidental faults:* As mentioned before an embedded system is a complex network of components. Therefore, a fault in a component can create a ripple of events throughout the network. This makes recovery for embedded systems extremely challenging. A causality graph for an embedded system is a graph where the nodes are events and edges are the causality between events ($e \rightarrow e'$ means that event e can cause event e'). Techniques for discovering a causality graph of an embedded is essentially for recovering from faults. Essentially the effects of a fault can be determined from examining the causality graph. Techniques for constructing attack graphs [1, 5] and alert correlation [3, 4]

References

[1] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *ACM Conference on Computer and Communications Security*, 2002.

[2] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha. Towards automatic generation of vulnerability based signatures. In *IEEE Symposium on Security and Privacy*, pages 21–24, May 2006.

[3] F. Cuppens and A. Mige. Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy*, 2002.

[4] P. Ning, Y. Cui, and D. S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *ACM Conference on Computer and Communications Security*, 2002.

[5] O. Sheyner, J. W. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *IEEE Symposium on Security and Privacy*, 2002.

[6] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier. Shield: Vulnerability-driven network filters for preventing known vulnerability exploits. In *In the Proceedings of ACM SIGCOMM*, Portland, OR, August 2004.

On Software Protection in Embedded Systems

Jisoo Yang and Kang G. Shin
Department of Electrical Engineering and Computer Science
The University of Michigan
{jisooy,kgshin}@eecs.umich.edu

Abstract

We argue that the conventional privilege separation of a processor has inherent limitations in protecting software with higher security requirements, and hence, a new system of protection should be devised to overcome these limitations. To enable the new protection, an operating system needs to be restructured into two layers: the security kernel which implements the new protection system, and the management kernel which manages resources. The security kernel protects the applications even when the management kernel is compromised. The security kernel should be made very thin and simple, thus making it suitable for small devices like handsets and smart sensors & actuators.

Limitations of Current Software Protection

With increasing computation power and storage capacity, many embedded systems are adopting the paradigm of user/kernel separation of a processor [3] to provide better software protection and management. This protection paradigm is characterized by a complete separation of privilege. Code executing in user mode (i.e., applications) is prevented from performing sensitive operations, whereas code executing in kernel mode (i.e., operating system) is considered privileged and hence, given unlimited power.

This simple protection mechanism is effective in protecting the operating system (OS), but it does not serve well the security needs of user applications. In an embedded environment, where applications usually perform critical operations and carry sensitive data, the application must be protected as strongly as the OS. Although the OS provides certain protection to the applications, there are inherent limitations with the simple user/kernel separation and complete reliance on the OS for the application protection.

First, there is no effective second-line of defense that applications can resort to in case the OS is compromised. Many applications need to protect secret/confidential information, but once an attacker seizes the control of the OS, it is very easy for the attacker to observe/steal the information. Any effort to further protect the information will be futile as the attacker can exploit the OS's power to subvert, reverse-engineer, or simply disable the protection mechanism.

Second, verifying the correctness of an OS is becoming intractable as its size and functionality continuously grow—even in an embedded environment—to meet the increasing demand for more features. Today's mobile phones, for instance, require some features comparable to those of PCs. Unfortunately, it is difficult to reduce the growing OS verification need due to the coarse-grained user/kernel separation, where every privileged code has unlimited power and thus, is subject to verification.

Third, the user/kernel separation generates trust dependencies among software components, which do not generally correspond to the relations of the component providers. This mismatch incurs assurance overhead and generates unwarranted conflicts of interest. For example, user applications must trust the OS. To trust the OS, the application providers need assurance of the OS's trustworthiness. For the assurance, a complete and unbiased validation of the OS is necessary, but doing so generally goes against the interest of the OS provider due to the cost of validation and the risk of exposing the system to others.

Challenges in Designing New Application Protection

We argue for the need of another system of protection that can deal with the above limitations. The new system should be able to protect applications even in the case of OS compromises, reduce the size of code required for verification, and break the trust dependencies between software components. To design and implement such a protection system, we must overcome the following challenges.

The first challenge is to define an appropriate threat model for user applications. We need to identify the security properties that the applications/users want for protecting their information/data, so that the new protection mechanism can preserve themselves even if the OS were compromised. Unfortunately, our problem is not in the secure communication domain, and thus, it is difficult to borrow familiar security properties from that domain. Also, we need to avoid over-protection for simplicity. Therefore, we need a threat model that represents the problem domain and captures essential security needs of the applications.

The second challenge is to preserve the OS's usual management power. With the new protection, however, the OS is restricted somewhat; the new protection system enforces certain rules and the OS is prevented from performing actions against the rules. However, the restriction should not obstruct the OS from performing a legitimate management job.

The third challenge is to find an implementation that is small and simple to verify. With the new protection, the OS can be verified less stringently, since applications can still be protected even when the OS fails (as a result of its compromise). However, the mechanism that implements the new protection should be fully trusted, and hence, the correctness of the implemented protection is critical to the security of the entire system.

Security Kernel

The new protection requires a different OS arrangement which consists of two layers of kernel: security and management kernels. Running with complete privilege, the security kernel is a very thin layer that only implements the new protection system. It must be fully trusted and must thus be rigorously verified. The management kernel, responsible for resource management and scheduling, is a restricted version of a conventional OS. As it runs on top of the security kernel, applications are still protected from any compromise in the management kernel. In this sense, it does not have to be trusted and verified. Both security and management kernels are protected from user applications by the traditional user/kernel separation.

Since it is small and simple enough, the security kernel can be realized entirely with hardware. Equipped with a circuitry that implements the protection logic, a processor can extend its ISA to expose a programming interface for the management kernel and user applications.

A software-only solution is also possible by using virtualization techniques. A virtual machine monitor (VMM) is capable of not only running multiple OSes, but also realizing hardware extensions or implementing system services without actually changing the real machine [1]. The VMM is more privileged than the OS and its perimeter is safe. Therefore, the security kernel can be implemented inside of the VMM.

Although we can implement the security kernel by modifying a full-fledged VMM such as Xen [2], a full VMM is not necessary as we do not have to run multiple OSes. Instead, a lightweight security kernel is preferred only by using the techniques and constructs required to enable the hardware extensions and to safeguard the VMM's perimeter.

Impact and Outlook

The new software protection system will make long-term impacts since it relaxes many assumptions currently made when software systems are composed. For instance, the management OS is no longer assumed to be trusted, thus creating opportunities for design of ambitious distributed systems which were risky under the assumption of trusted OS. Also, existing software systems can be retroactively redesigned to exploit the enlarged design space, thus making them more reliable with minimal additional effort.

Conclusion

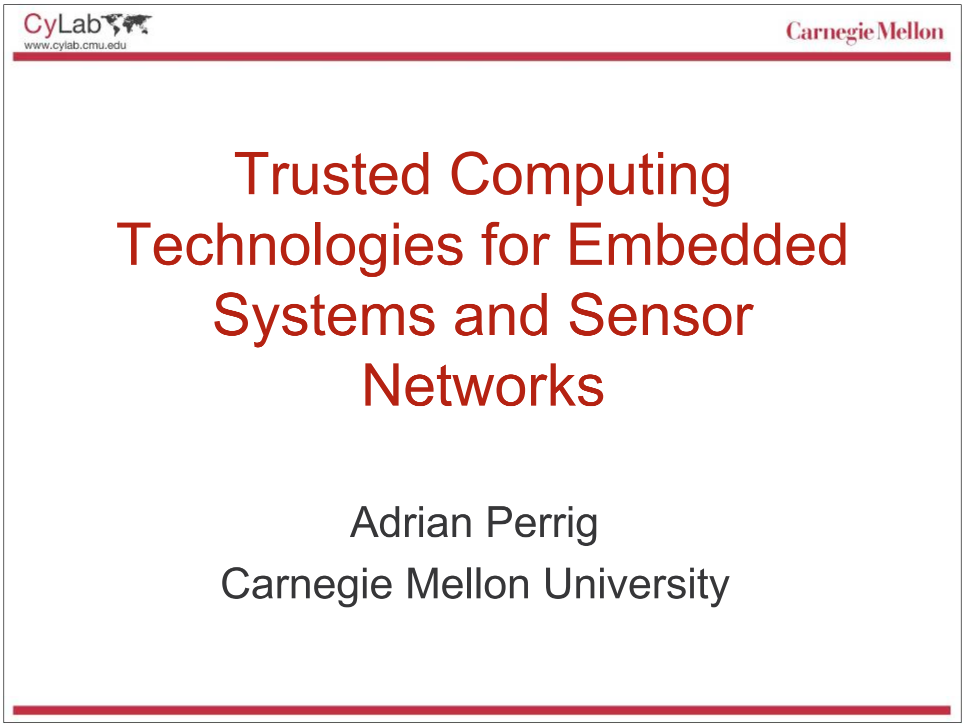
The conventional user/kernel separation is not sufficient to meet the growing demand for software protection in embedded systems. We argue for the need of a new protection mechanism that can protect user applications, lessen verification overhead, and break trust dependencies. The new protection is enforced by a 'security kernel' which can be realized as a lightweight software layer using virtualization techniques, making it suitable for small devices and embedded systems, such as handsets and smart sensors & actuators.

References

[1] Peter M. Chen and Brian D. Noble. When virtual is better than real. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, May 2001.

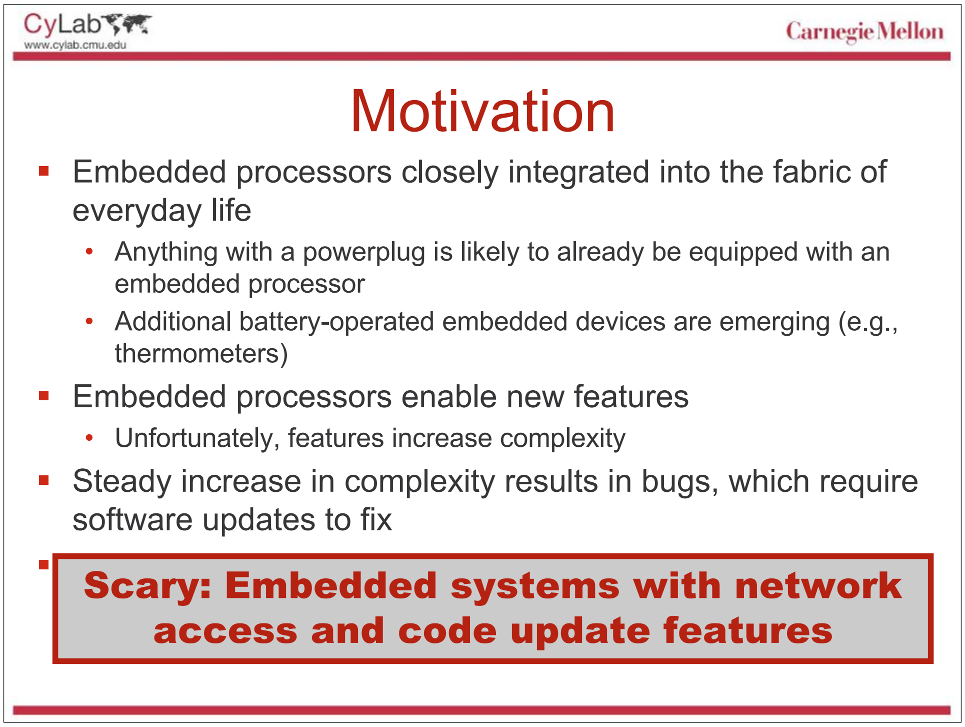
[2] Boris Dragovic, Keir Fraser, Steve Hand, Tim Harris, Alex Ho, Ian Pratt, Andrew Warfield, Paul Barham, and Rolf Neugebauer. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, Oct 2003.

[3] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, Sep 1975.



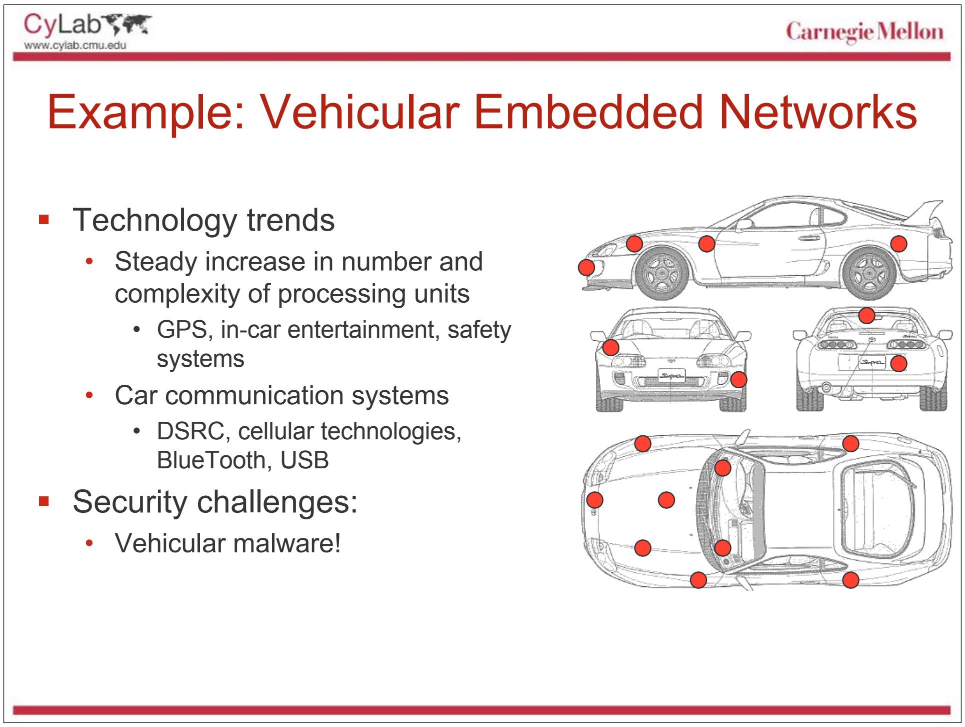
Trusted Computing Technologies for Embedded Systems and Sensor Networks

Adrian Perrig
Carnegie Mellon University



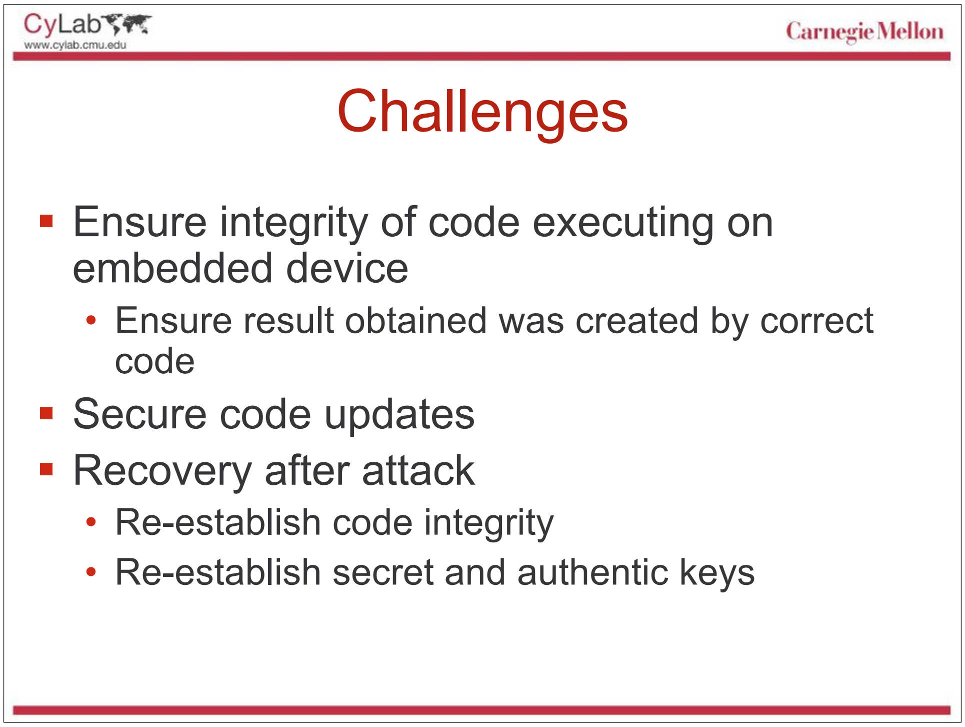
Motivation

- Embedded processors closely integrated into the fabric of everyday life
  - Anything with a powerplug is likely to already be equipped with an embedded processor
  - Additional battery-operated embedded devices are emerging (e.g., thermometers)
- Embedded processors enable new features
  - Unfortunately, features increase complexity
- Steady increase in complexity results in bugs, which require software updates to fix
- **Scary: Embedded systems with network access and code update features**



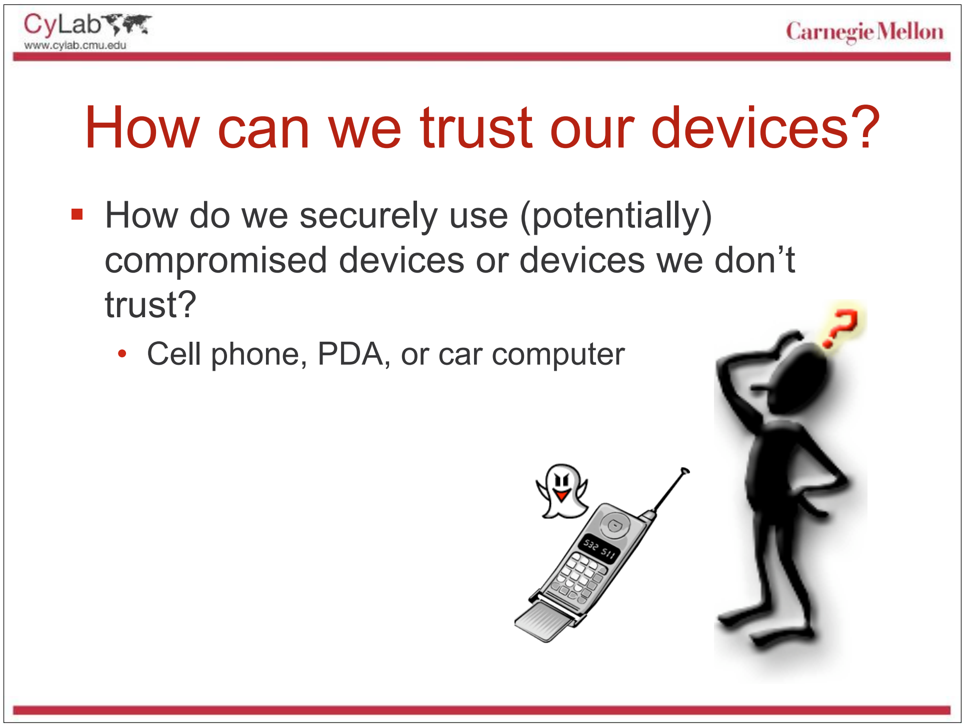
Example: Vehicular Embedded Networks

- Technology trends
  - Steady increase in number and complexity of processing units
    - GPS, in-car entertainment, safety systems
  - Car communication systems
    - DSRC, cellular technologies, BlueTooth, USB
- Security challenges:
  - Vehicular malware!



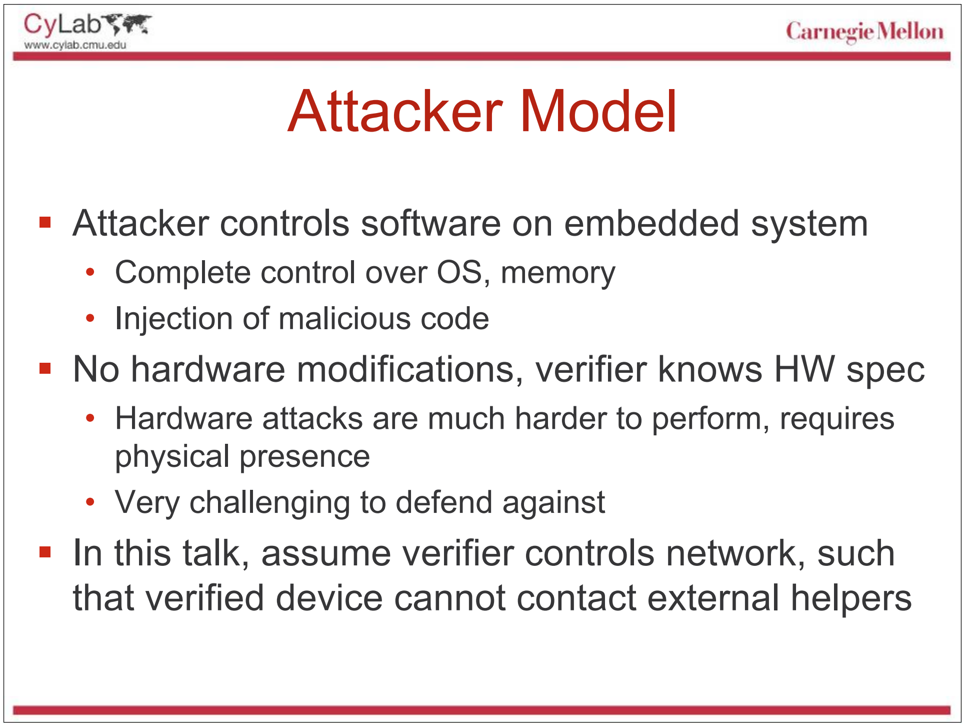
Challenges

- Ensure integrity of code executing on embedded device
  - Ensure result obtained was created by correct code
- Secure code updates
- Recovery after attack
  - Re-establish code integrity
  - Re-establish secret and authentic keys



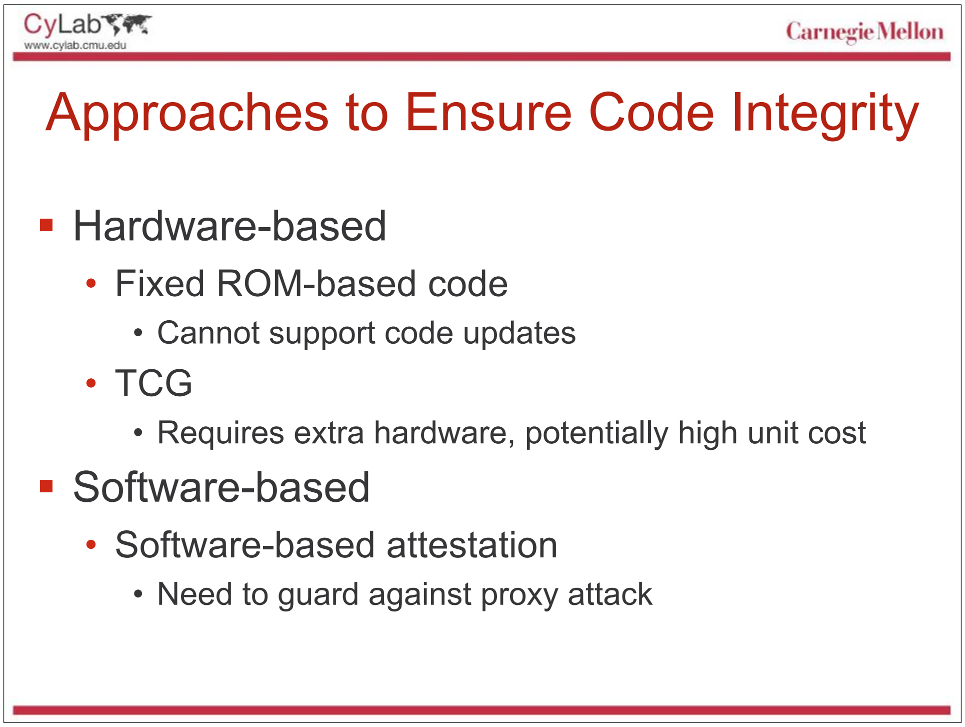
How can we trust our devices?

- How do we securely use (potentially) compromised devices or devices we don't trust?
  - Cell phone, PDA, or car computer



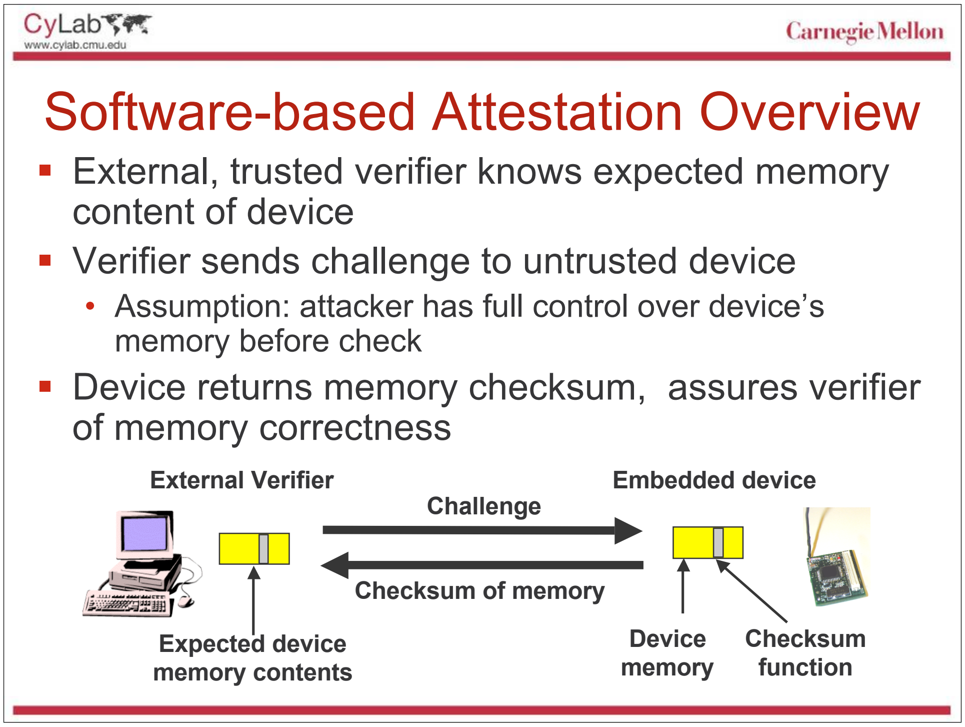
Attacker Model

- Attacker controls software on embedded system
  - Complete control over OS, memory
  - Injection of malicious code
- No hardware modifications, verifier knows HW spec
  - Hardware attacks are much harder to perform, requires physical presence
  - Very challenging to defend against
- In this talk, assume verifier controls network, such that verified device cannot contact external helpers



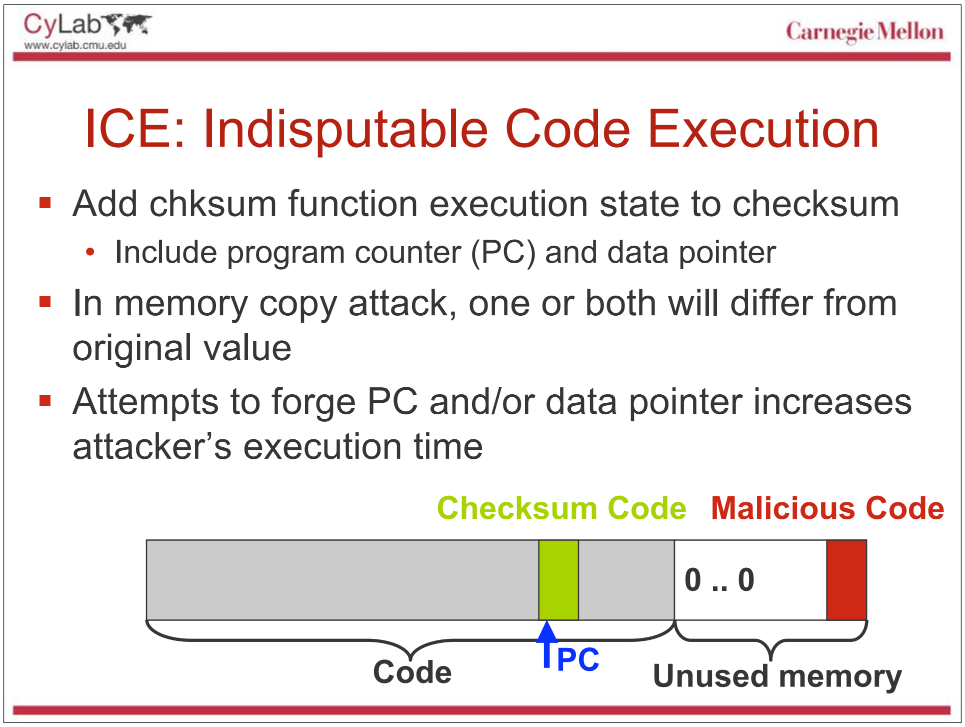
Approaches to Ensure Code Integrity

- Hardware-based
  - Fixed ROM-based code
    - Cannot support code updates
  - TCG
    - Requires extra hardware, potentially high unit cost
- Software-based
  - Software-based attestation
    - Need to guard against proxy attack



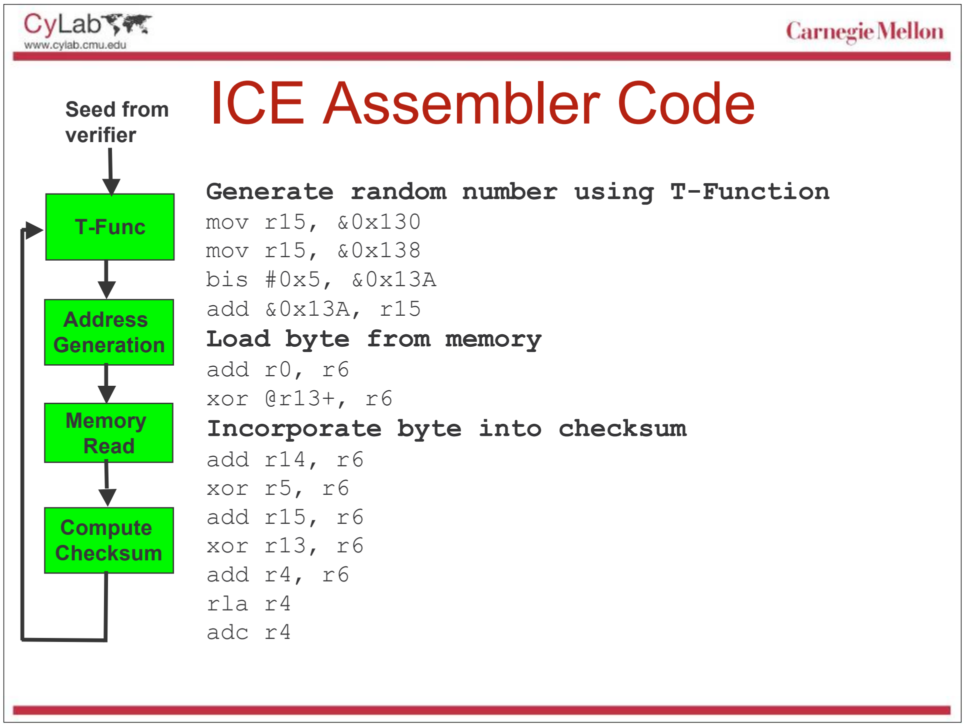
Software-based Attestation Overview

- External, trusted verifier knows expected memory content of device
- Verifier sends challenge to untrusted device
  - Assumption: attacker has full control over device's memory before check
- Device returns memory checksum, assures verifier of memory correctness



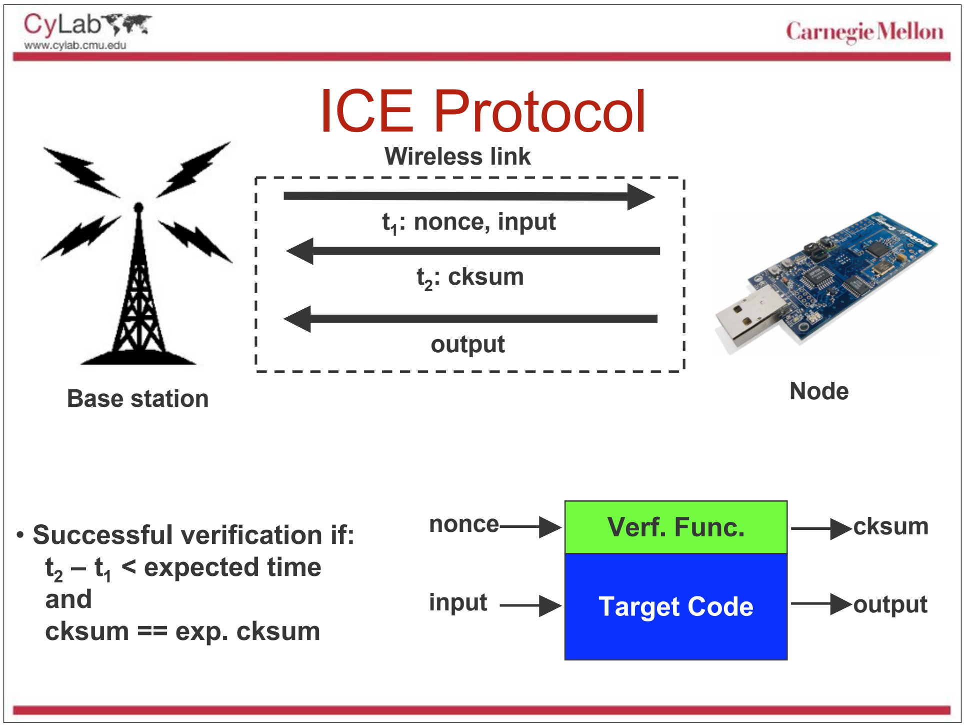
ICE: Indisputable Code Execution

- Add chksum function execution state to checksum
  - Include program counter (PC) and data pointer
- In memory copy attack, one or both will differ from original value
- Attempts to forge PC and/or data pointer increases attacker's execution time



ICE Assembler Code

```
Generate random number using T-Function
mov r15, &0x130
mov r15, &0x138
bis #0x5, &0x13A
add &0x13A, r15
Load byte from memory
add r0, r6
xor @r13+, r6
Incorporate byte into checksum
add r14, r6
xor r5, r6
add r15, r6
xor r13, r6
add r4, r6
rla r4
adc r4
```



ICE Protocol

Wireless link

t_1: nonce, input

t_2: cksum

output

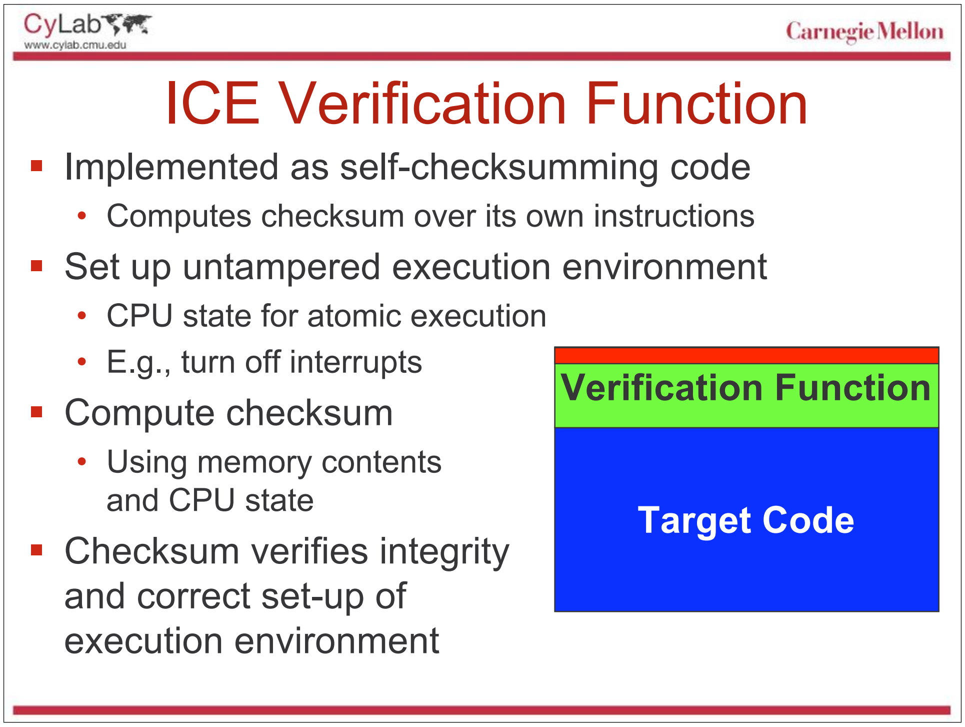
Base station

Node

- Successful verification if:
  $t_2 - t_1$ < expected time
  and
  cksum == exp. cksum

nonce → Verf. Func. → cksum

input → Target Code → output



ICE Verification Function

- Implemented as self-checksumming code
  - Computes checksum over its own instructions
- Set up untampered execution environment
  - CPU state for atomic execution
  - E.g., turn off interrupts
- Compute checksum
  - Using memory contents and CPU state
- Checksum verifies integrity and correct set-up of execution environment

Verification Function

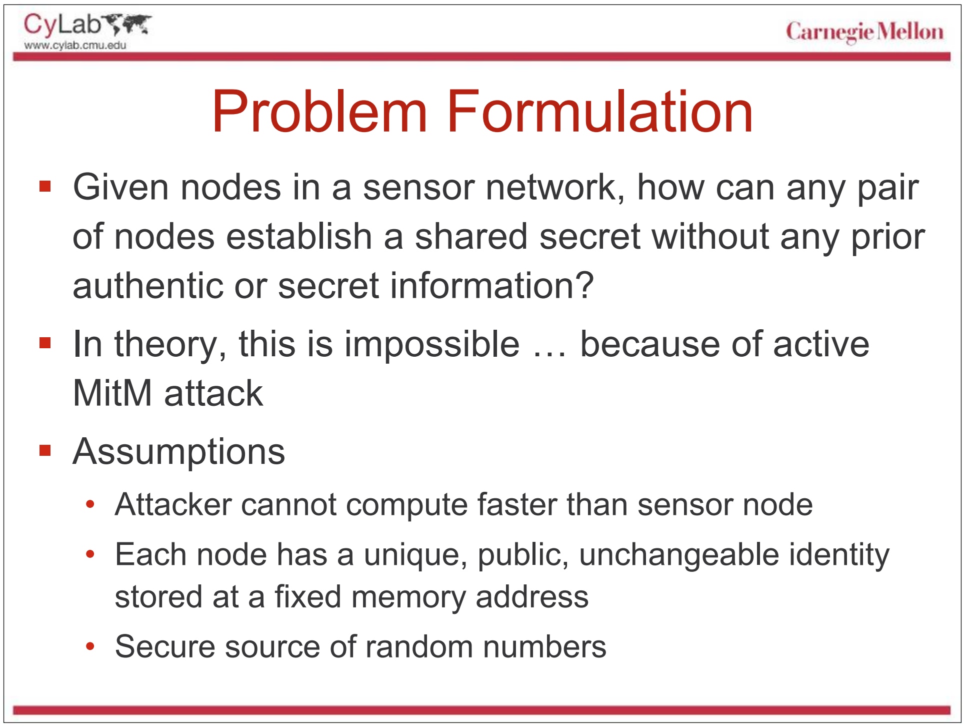
Target Code

ICE Properties

- Given target code T, verifier obtains property that sensor node S correctly executes T, untampered by any other (malicious) code potentially present on S
- By incorporating node ID into checksum computation, we can authenticate response

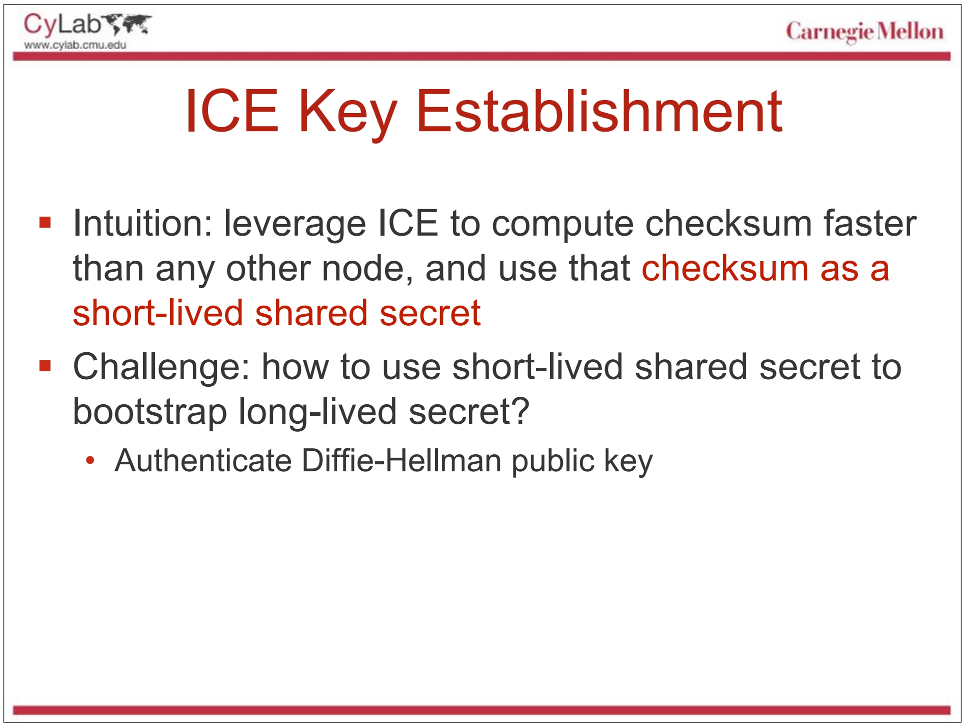
Key Establishment

- How to establish a shared secret?
  - Attacker may know entire memory contents of a newly shipped node
  - After a node has been compromised, attacker may have altered authentic public keys or knows secret keys
  - Without authentication Diffie-Hellman protocol is vulnerable to man-in-the-middle attack:
    - A → B: g^a mod p
    - B → A: g^b mod p



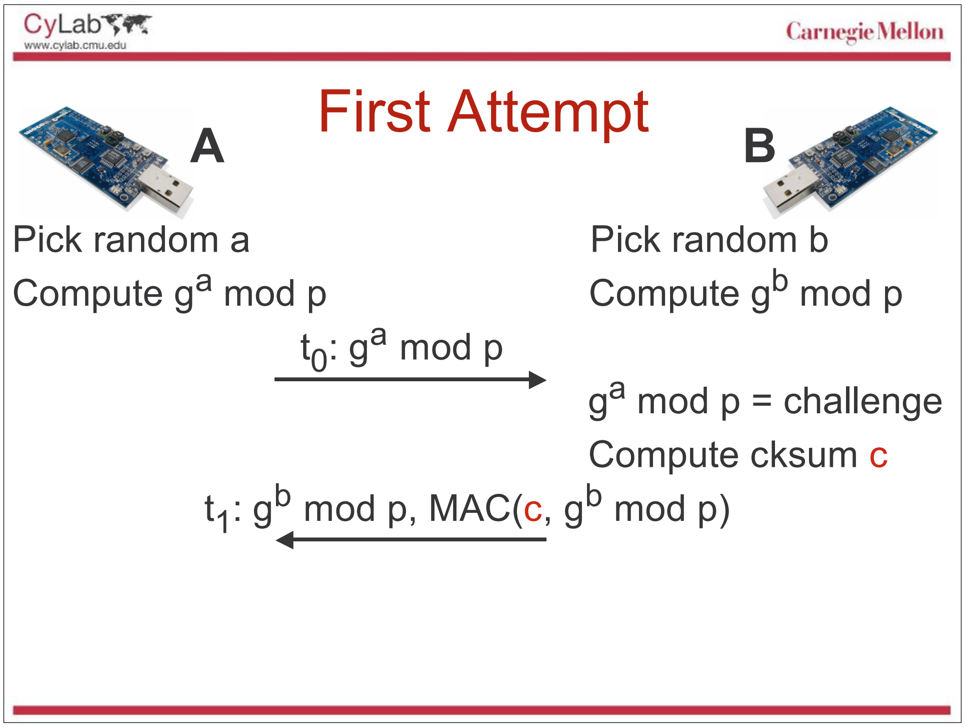
Problem Formulation

- Given nodes in a sensor network, how can any pair of nodes establish a shared secret without any prior authentic or secret information?
- In theory, this is impossible … because of active MitM attack
- Assumptions
  - Attacker cannot compute faster than sensor node
  - Each node has a unique, public, unchangeable identity stored at a fixed memory address
  - Secure source of random numbers



ICE Key Establishment

- Intuition: leverage ICE to compute checksum faster than any other node, and use that checksum as a short-lived shared secret
- Challenge: how to use short-lived shared secret to bootstrap long-lived secret?
  - Authenticate Diffie-Hellman public key



First Attempt

A

B

Pick random a

Compute g^a mod p

Pick random b

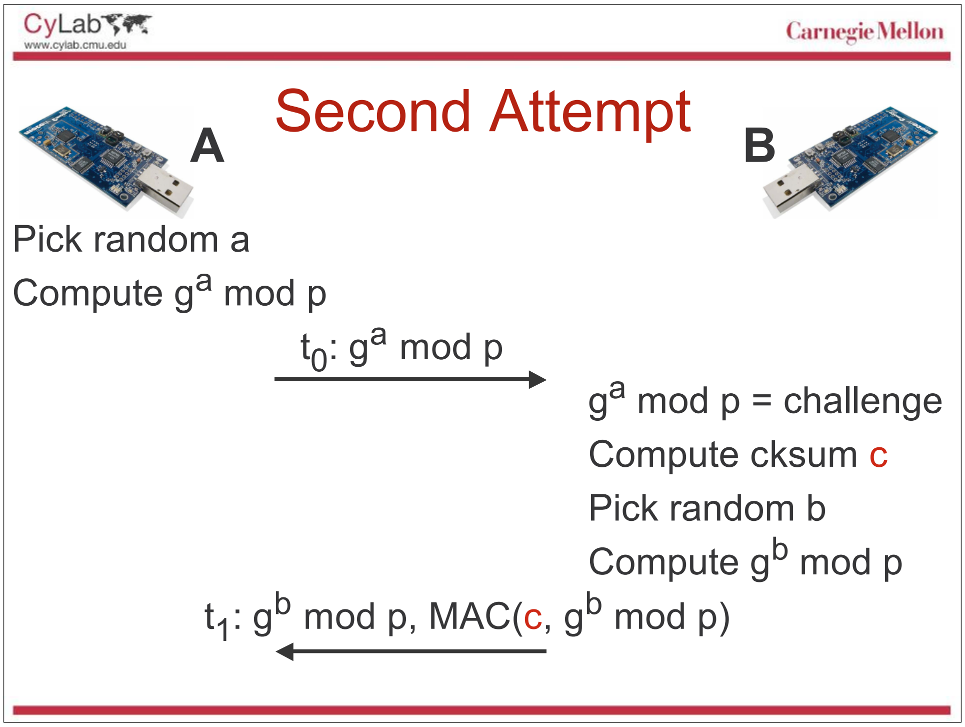
Compute g^b mod p

t_0: g^a mod p →

g^a mod p = challenge

Compute cksum c

t_1: g^b mod p, MAC(c, g^b mod p) ←



Second Attempt

A

B

Pick random a

Compute g^a mod p

t_0: g^a mod p →

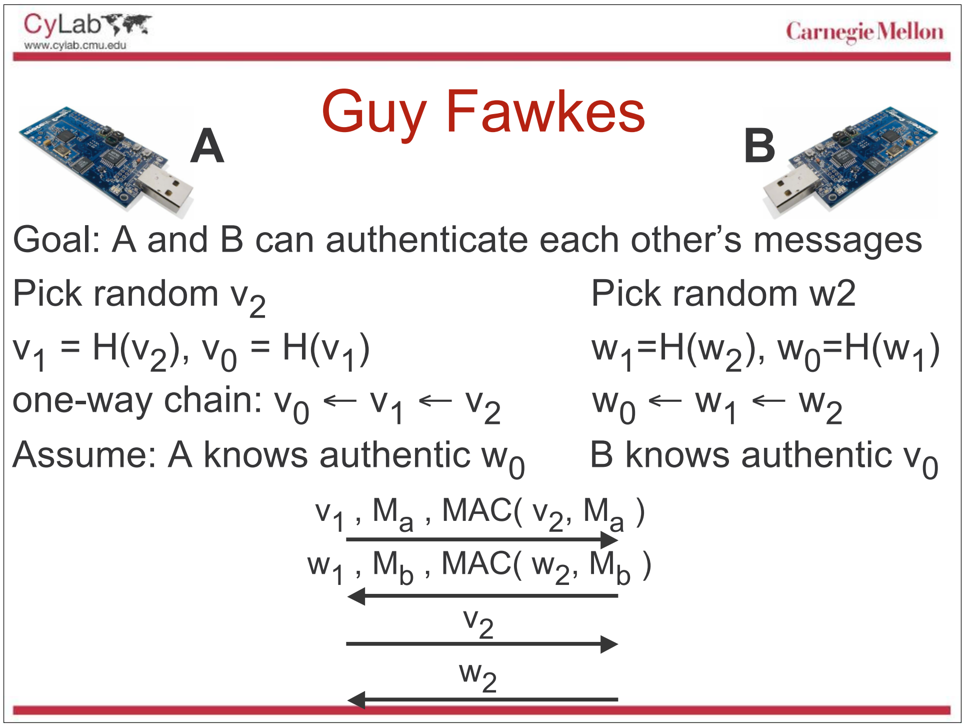
g^a mod p = challenge

Compute cksum c

Pick random b

Compute g^b mod p

t_1: g^b mod p, MAC(c, g^b mod p) ←



Guy Fawkes

A

B

Goal: A and B can authenticate each other's messages

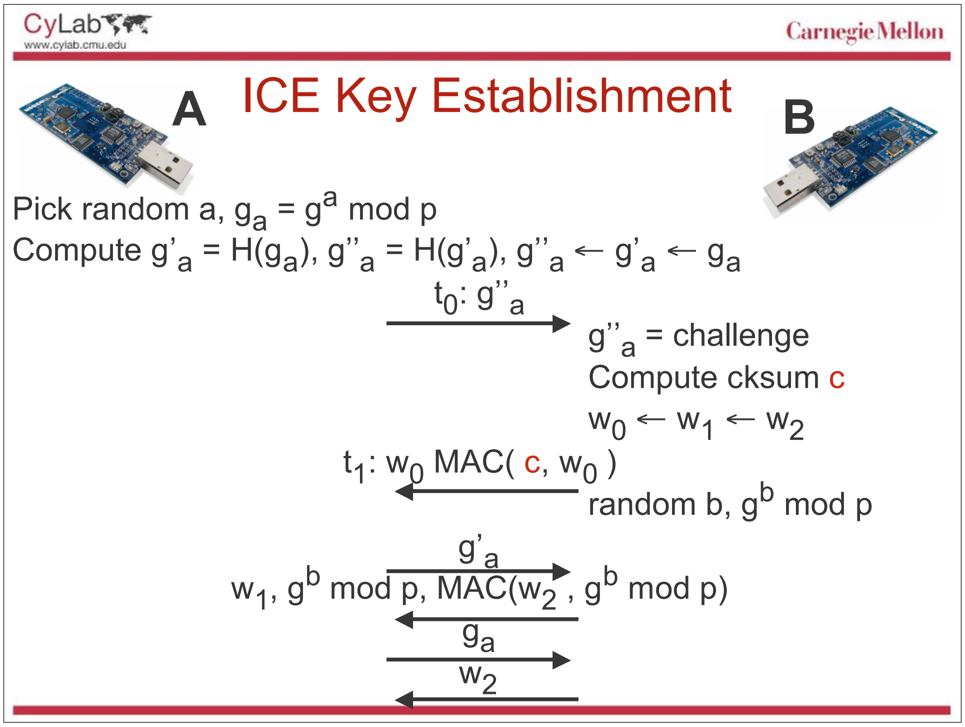
Pick random v_2	Pick random w2
$v_1 = H(v_2)$, $v_0 = H(v_1)$	$w_1=H(w_2)$, $w_0=H(w_1)$
one-way chain: $v_0 \leftarrow v_1 \leftarrow v_2$	$w_0 \leftarrow w_1 \leftarrow w_2$
Assume: A knows authentic w_0	B knows authentic v_0

v_1 , M_a , MAC(v_2, M_a) →

w_1 , M_b , MAC(w_2, M_b) ←

v_2 →

w_2 ←



ICE Key Establishment

A

B

Pick random a, $g_a = g^a$ mod p

Compute $g'_a = H(g_a)$, $g''_a = H(g'_a)$, $g''_a \leftarrow g'_a \leftarrow g_a$

t_0: g''_a →

g''_a = challenge

Compute cksum c

$w_0 \leftarrow w_1 \leftarrow w_2$

t_1: w_0 MAC(c, w_0) ←

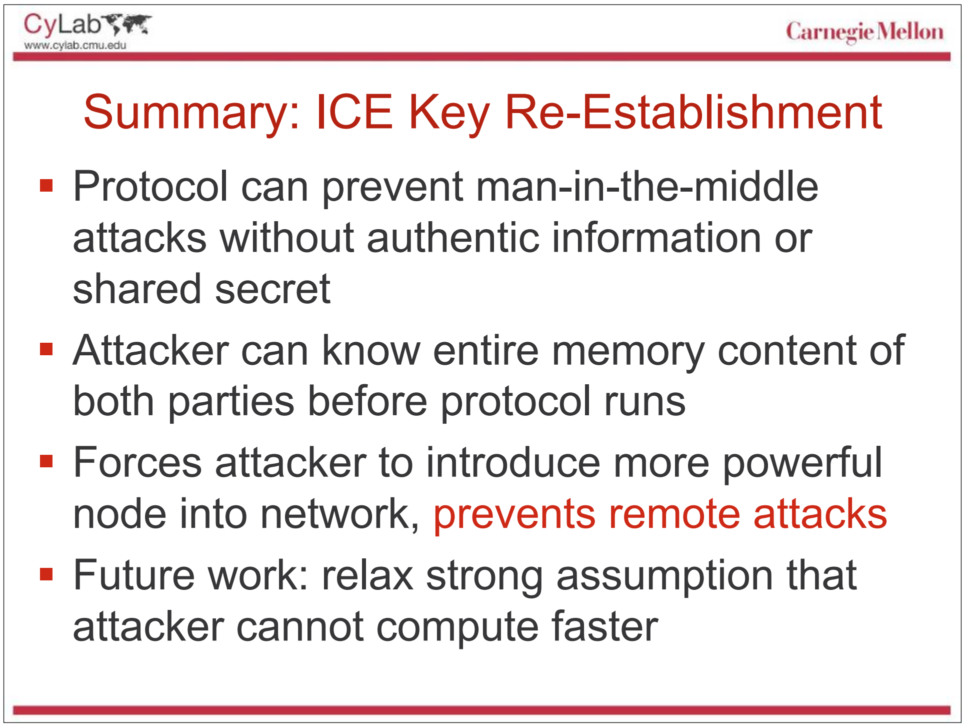
random b, g^b mod p

g'_a →

w_1, g^b mod p, MAC(w_2 , g^b mod p) ←

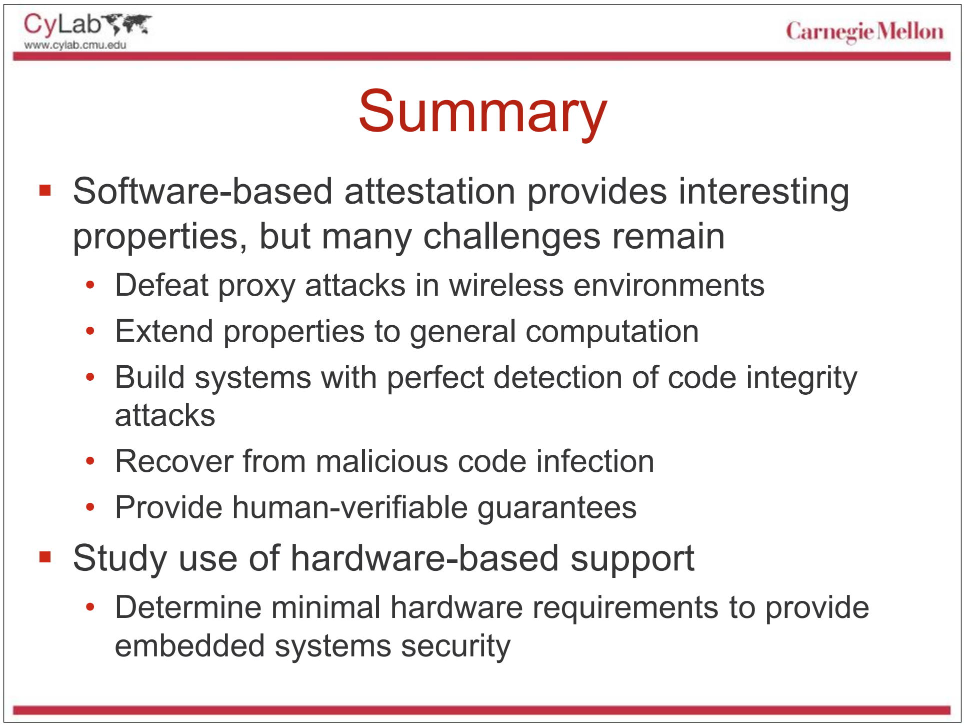
g_a →

w_2 ←



Summary: ICE Key Re-Establishment

- Protocol can prevent man-in-the-middle attacks without authentic information or shared secret
- Attacker can know entire memory content of both parties before protocol runs
- Forces attacker to introduce more powerful node into network, prevents remote attacks
- Future work: relax strong assumption that attacker cannot compute faster



Summary

- Software-based attestation provides interesting properties, but many challenges remain
  - Defeat proxy attacks in wireless environments
  - Extend properties to general computation
  - Build systems with perfect detection of code integrity attacks
  - Recover from malicious code infection
  - Provide human-verifiable guarantees
- Study use of hardware-based support
  - Determine minimal hardware requirements to provide embedded systems security

Software Security Issues in Embedded Systems

Somesh Jha
University of Wisconsin

Software Security

- Vulnerability Assessment
  - Analysis tools for discovering vulnerabilities in source code and binaries
- Automated Signature Generation
  - Generating signatures that filter our malicious inputs
- Malicious Code Detection
  - Detecting whether a binary has malicious behavior

Embedded Systems

- Increasingly used in critical sectors
  - Defense, medical, power, …
- Malicious and accidental failures can have dire consequences
- Embedded systems are not “all hardware”
  - They have software too☺
- Autonomous nature

Dynamic and Configurable Environment

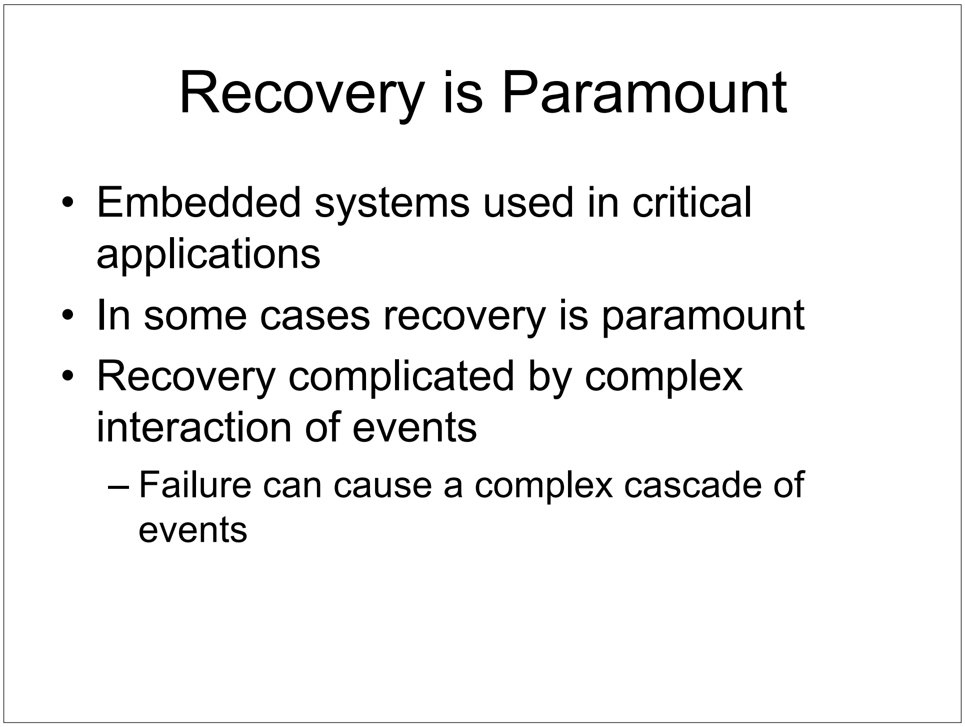
- Embedded systems are highly configurable
  - They have to work in many different scenarios
- Environment is highly dynamic
  - Think about embedded systems in a battlefield
  - Embedded system in a vehicle

Changing Functional Requirements

- Functional requirements of embedded systems change over time
- Embedded system deployed in a battlefield
  - Functional requirements change with mission

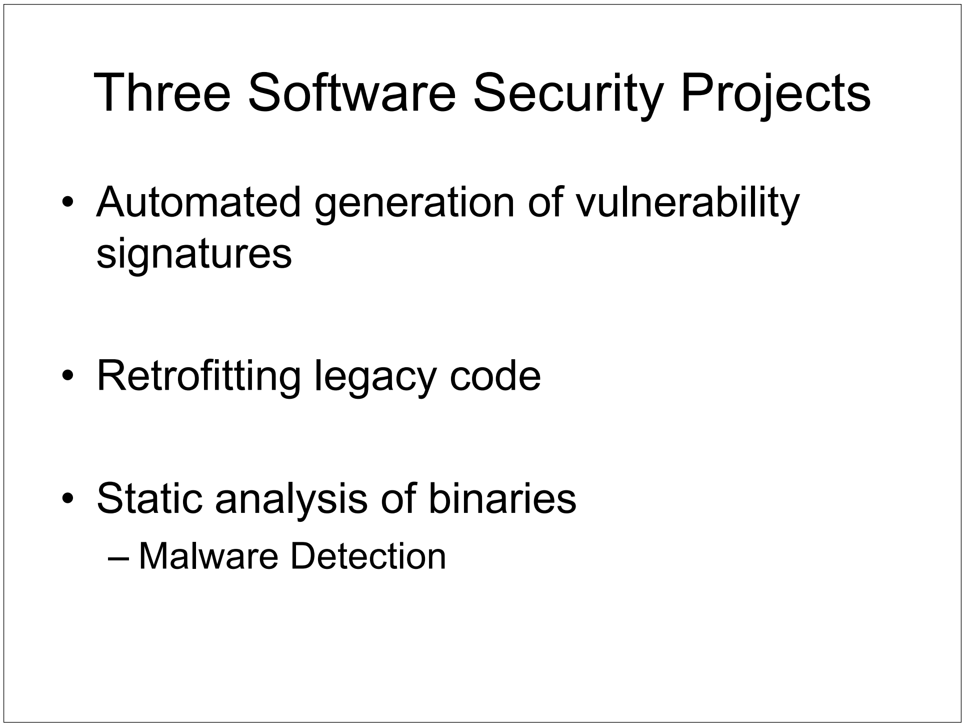
Interconnected Network of Components

- Embedded system are of a complex network of components
- Components might be hardware or software
- Source code might be available for some components
- COTS components (only binary available)
- Failure can create cascading events



Recovery is Paramount

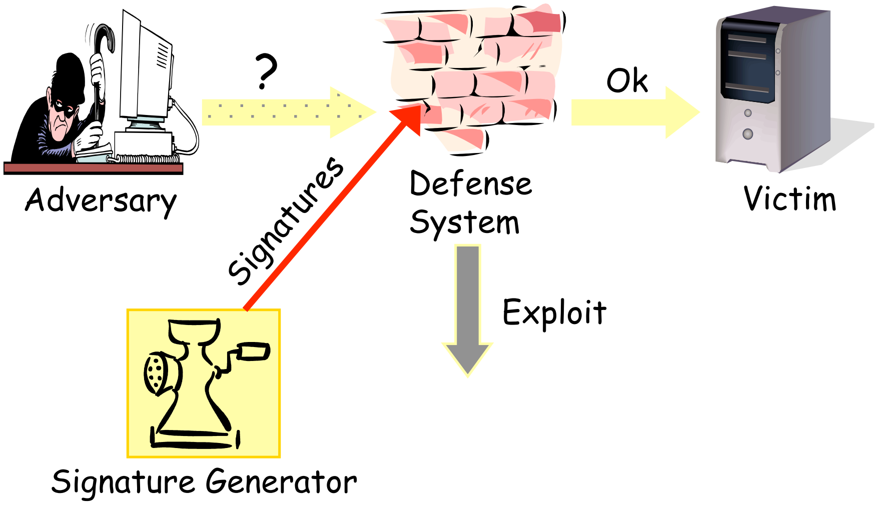
- Embedded systems used in critical applications
- In some cases recovery is paramount
- Recovery complicated by complex interaction of events
  - Failure can cause a complex cascade of events

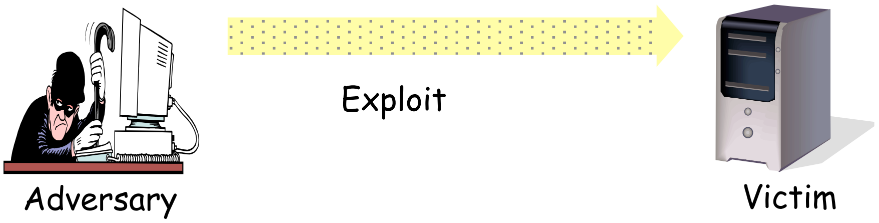


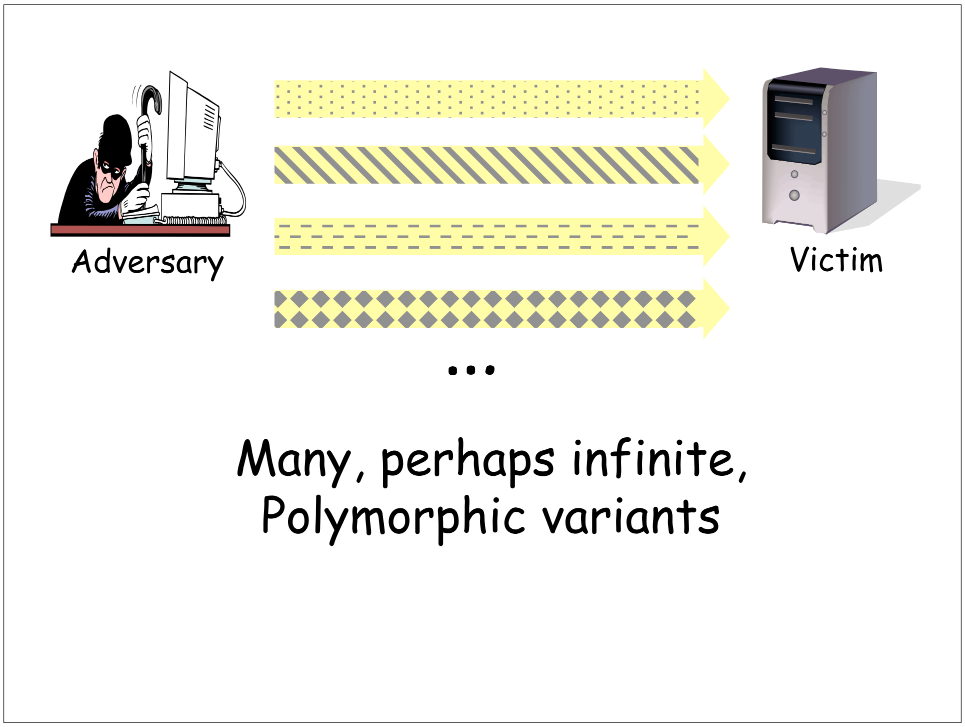
Three Software Security Projects

- Automated generation of vulnerability signatures
- Retrofitting legacy code
- Static analysis of binaries
  - Malware Detection

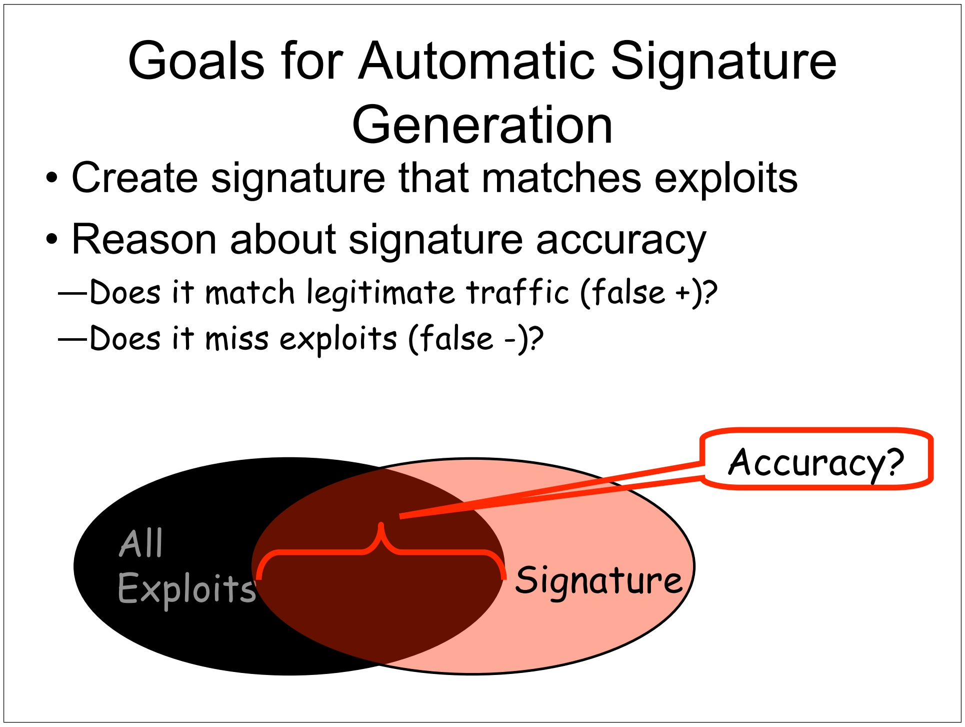
Motivating Scenario for Automatic Signature Generation







Many, perhaps infinite,
Polymorphic variants



Goals for Automatic Signature Generation

- Create signature that matches exploits
- Reason about signature accuracy
  - —Does it match legitimate traffic (false +)?
  - —Does it miss exploits (false -)?

Our Contribution:
A Language-Centric Approach

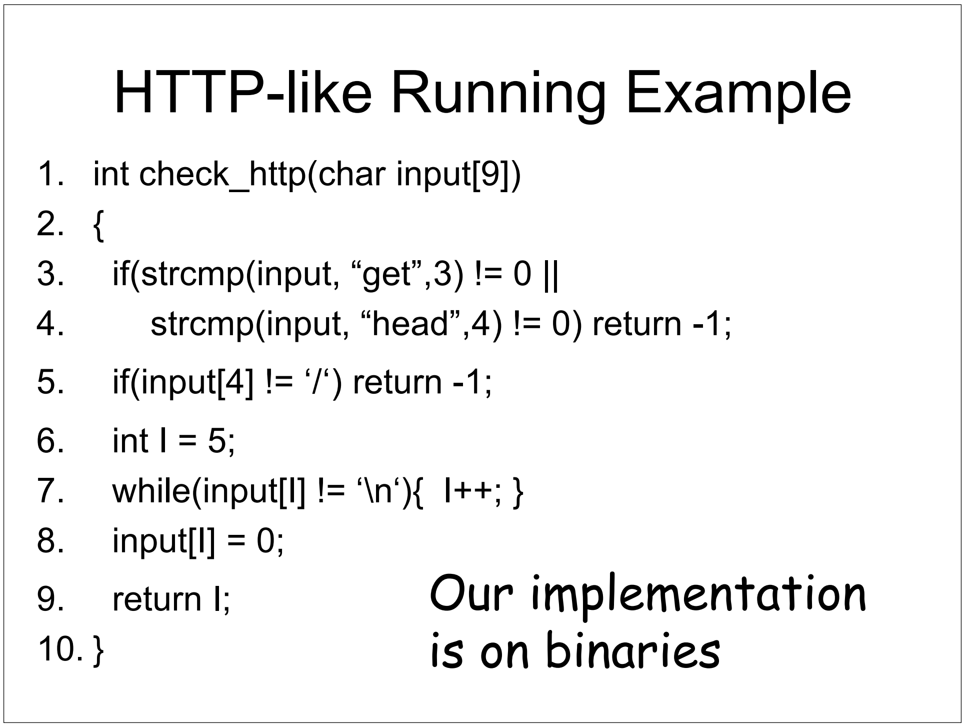
- Focus on the language of the vulnerability
  - ➡ Reason about signature via language
  - ➡ Language captures all exploits
- New methods for Automatic vulnerability signature creation
  - ➡Opens doors to PL techniques

Language of a Particular Vulnerability

- A vulnerability is defined by:

1. What – The Vulnerability Condition:
   Necessary conditions to violate safety
2. Where – The Vulnerability Point:
   Location vulnerability condition first satisfied

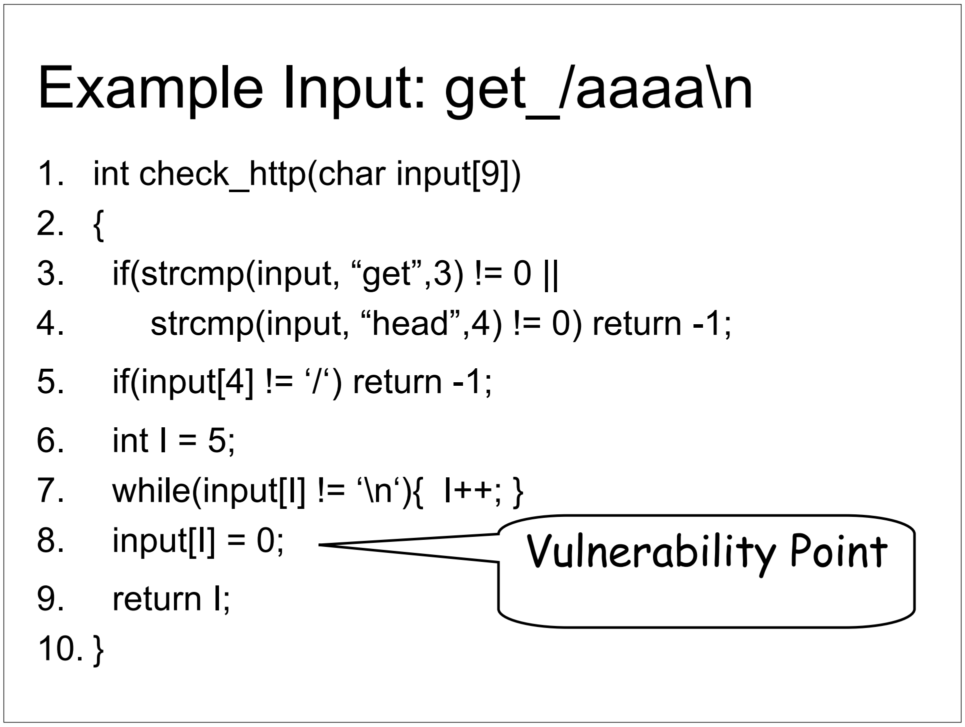
The Vulnerability Language is all input strings reaching the vulnerability point meeting the vulnerability condition.



HTTP-like Running Example

```
int check_http(char input[9])
{
  if(strcmp(input, “get”,3) != 0 ||
     strcmp(input, “head”,4) != 0) return -1;
  if(input[4] != ‘/‘) return -1;
  int I = 5;
  while(input[I] != ‘\n‘){  I++; }
  input[I] = 0;
  return I;
}
```

Our implementation is on binaries



Example Input: get_/aaaa\n

```
int check_http(char input[9])
{
  if(strcmp(input, “get”,3) != 0 ||
     strcmp(input, “head”,4) != 0) return -1;
  if(input[4] != ‘/‘) return -1;
  int I = 5;
  while(input[I] != ‘\n‘){  I++; }
  input[I] = 0;
  return I;
}
```

Vulnerability Point

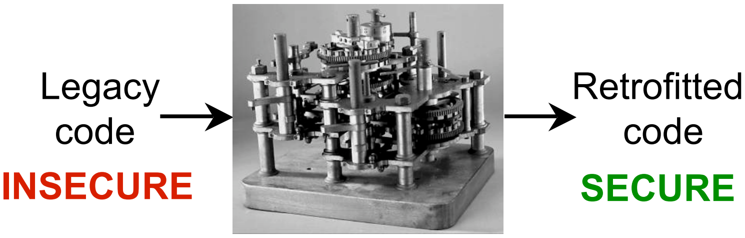
Example Input: get_/aaaa\n

```
int check_http(char input[9])
{
  if(strcmp(input, “get”,3) != 0 ||
     strcmp(input, “head”,4) != 0) return -1;
  if(input[4] != ‘/‘) return -1;
  int I = 5;
  while(input[I] != ‘\n‘){  I++; }
  input[I] = 0;
  return I;
}
```

Vulnerability Condition: I >= 9

Retrofitting legacy code

Need systematic techniques to retrofit legacy code for security



Legacy code → → Retrofitted code

INSECURE

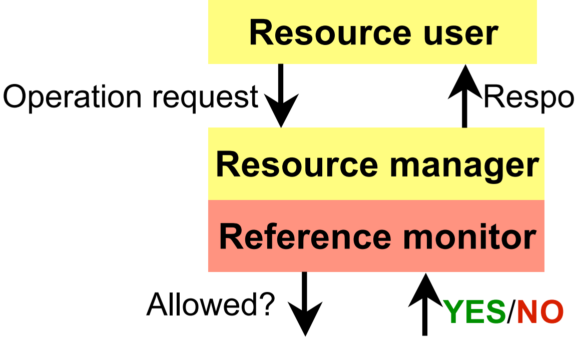
SECURE

Retrofitting legacy code

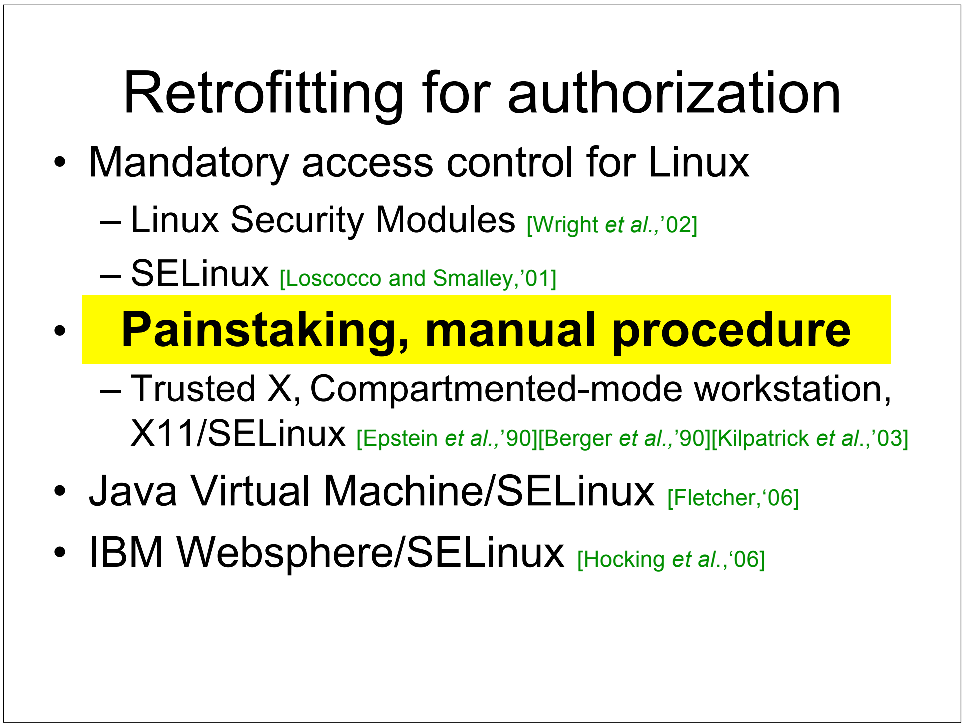
Need systematic techniques to retrofit legacy code for security

- Enforcing type safety
  - CCured [Necula *et al.* ’02]
- Partitioning for privilege separation
  - PrivTrans [Brumley and Song, ’04]
- Enforcing authorization policies

Enforcing authorization policies

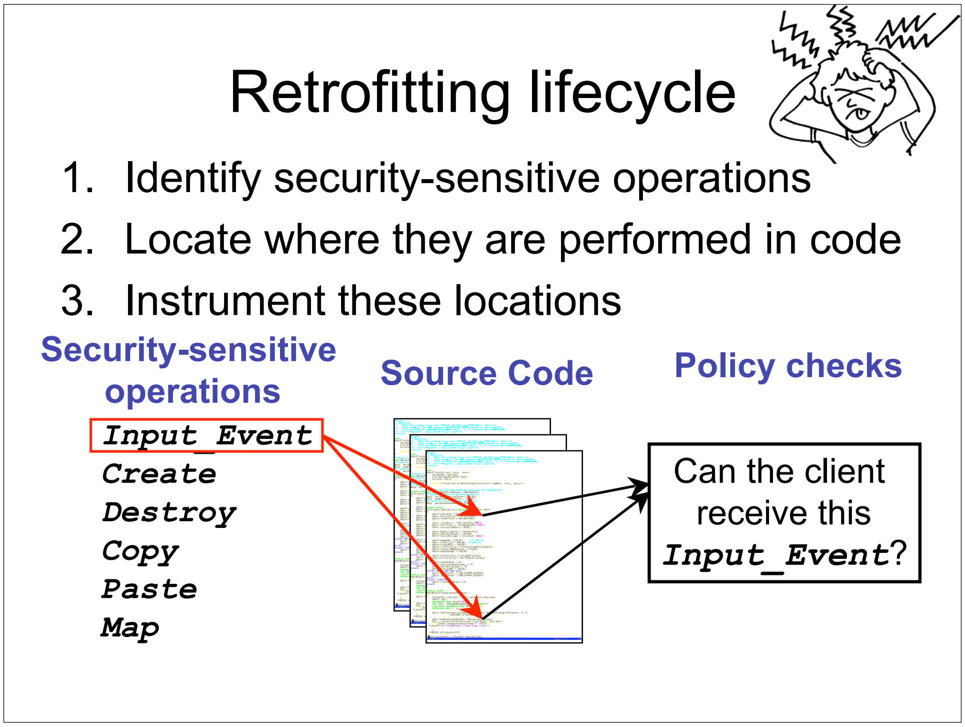


❬Alice, /etc/passwd, ***`File_Read`***❭



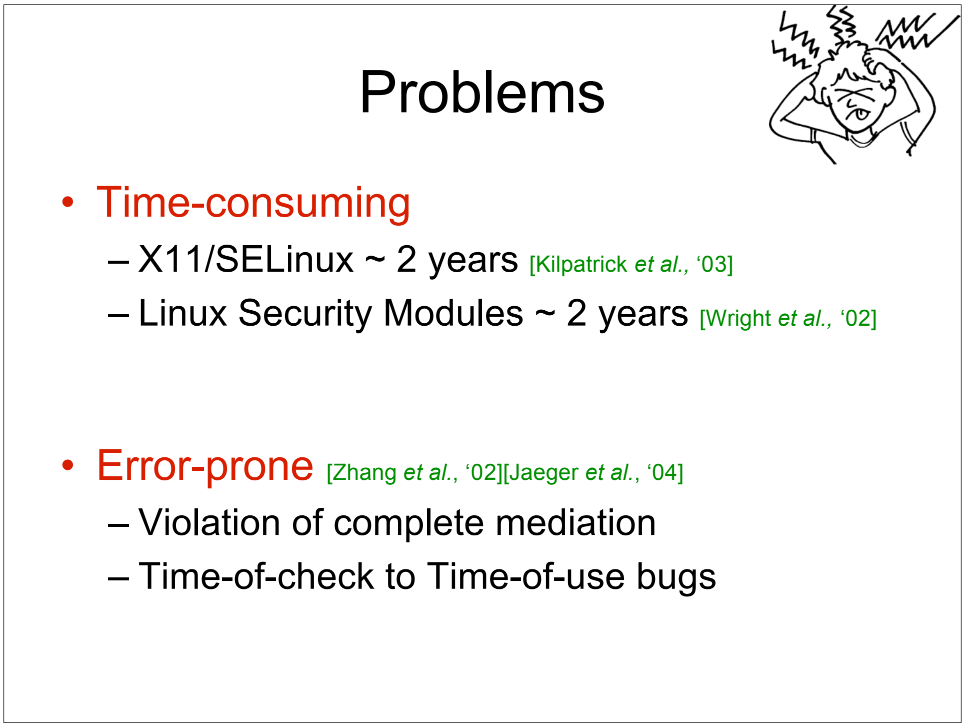
Retrofitting for authorization

- Mandatory access control for Linux
  - Linux Security Modules [Wright *et al.*,'02]
  - SELinux [Loscocco and Smalley,'01]
- **Painstaking, manual procedure**
  - Trusted X, Compartmented-mode workstation, X11/SELinux [Epstein *et al.*,'90][Berger *et al.*,'90][Kilpatrick *et al.*,'03]
- Java Virtual Machine/SELinux [Fletcher,'06]
- IBM Websphere/SELinux [Hocking *et al.*,'06]



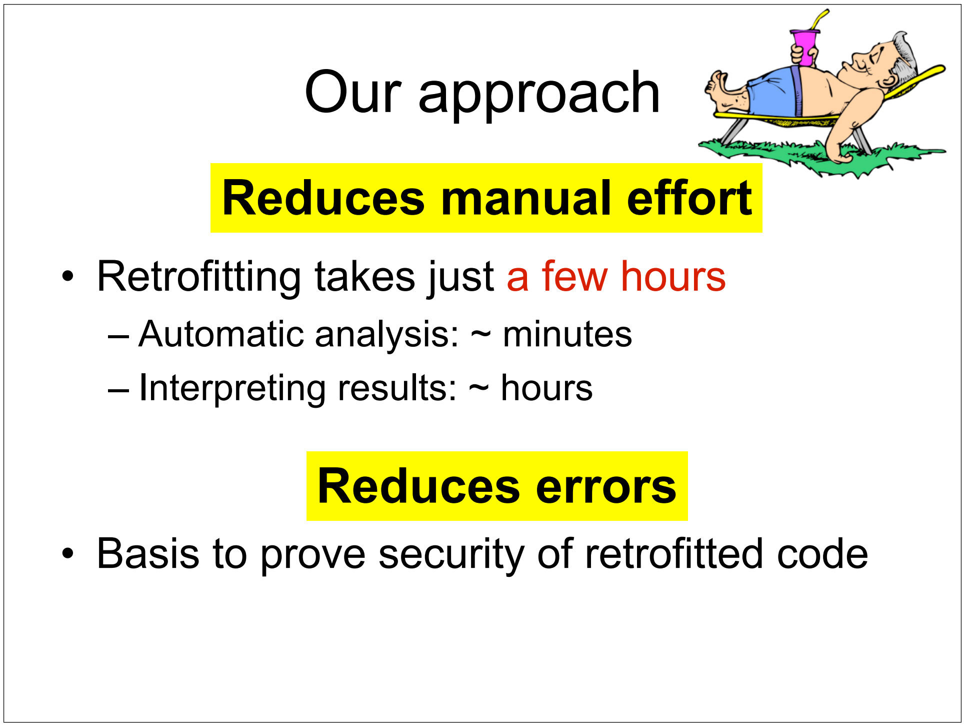
Retrofitting lifecycle

1. Identify security-sensitive operations
2. Locate where they are performed in code
3. Instrument these locations



Problems

- Time-consuming
  - X11/SELinux ~ 2 years [Kilpatrick *et al.*, '03]
  - Linux Security Modules ~ 2 years [Wright *et al.*, '02]
- Error-prone [Zhang *et al.*, '02][Jaeger *et al.*, '04]
  - Violation of complete mediation
  - Time-of-check to Time-of-use bugs



Our approach

Reduces manual effort

- Retrofitting takes just a few hours
  - Automatic analysis: ~ minutes
  - Interpreting results: ~ hours

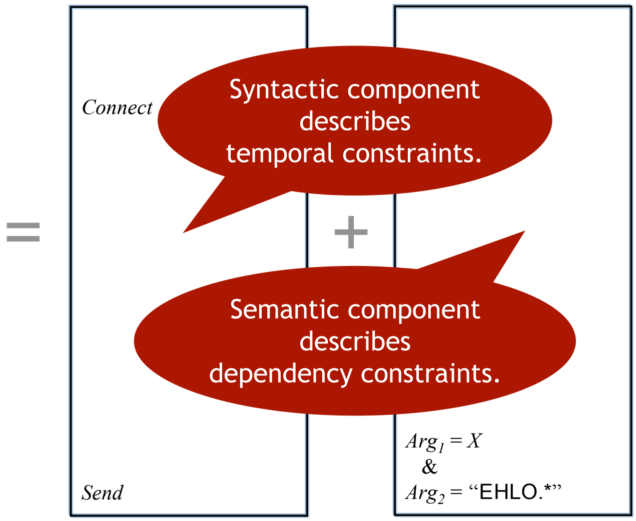
Reduces errors

- Basis to prove security of retrofitted code

Malspec: *Self-Propagation by Email*

```
push      10h
push      eax
push      edi
call      connect
push      esi
push      eax
push      [ebp+hMem]
call      wsprintfA
add       esp, 0Ch
push      [ebp+hMem]
call      lstrlenA
push      0
push      eax
push      [ebp+hMem]
push      ebx
push      eax
push      ecx
push      edi
call      send
```

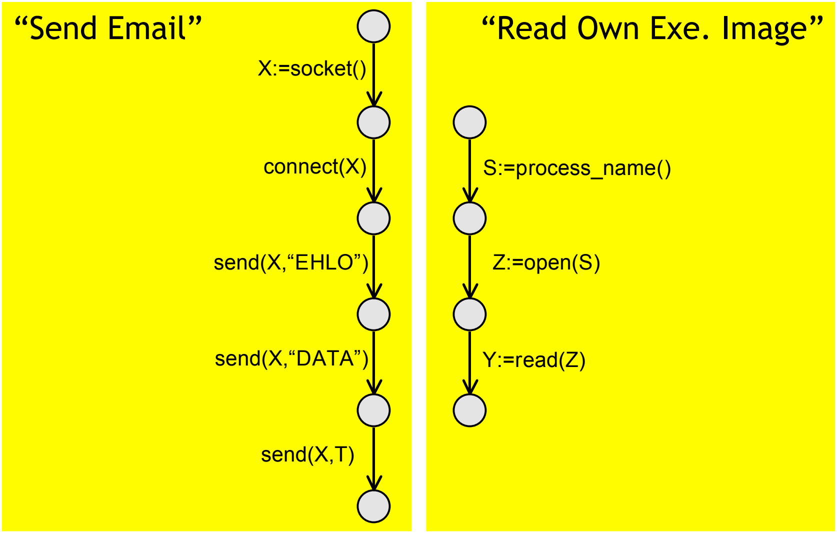
Netsky.B



Syntactic component describes temporal constraints.

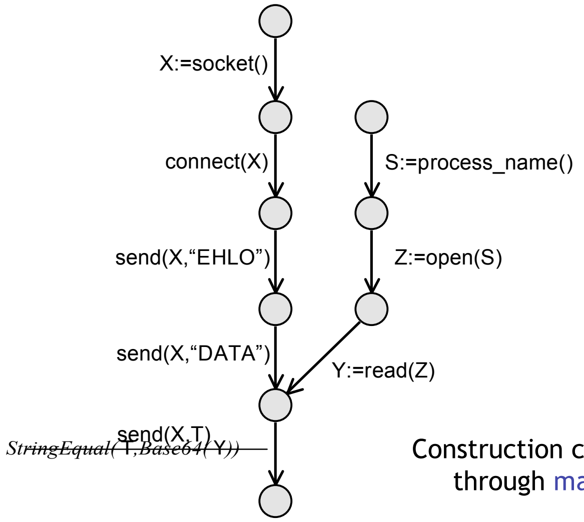
Semantic component describes dependency constraints.

Building a Real Malspec



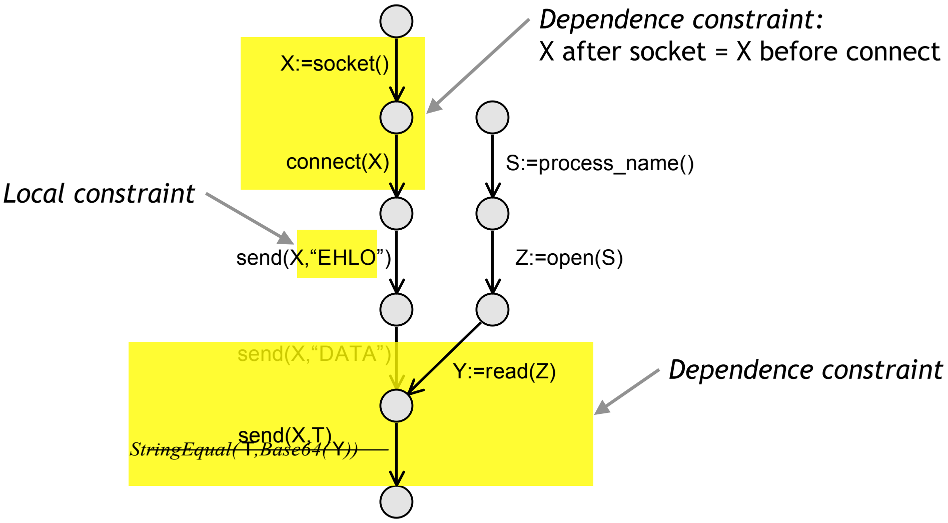
Malspec: *Self-Propagation by Email*

AND-OR graph



Construction can be automated through malspec mining.

Malspec Constraints

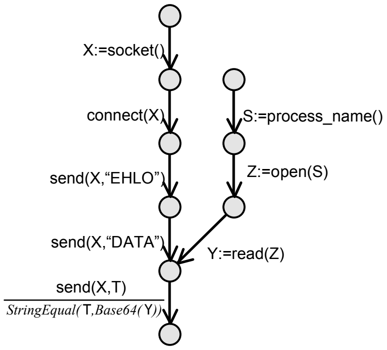


Dependence constraint:
X after socket = X before connect

Local constraint

Dependence constraint

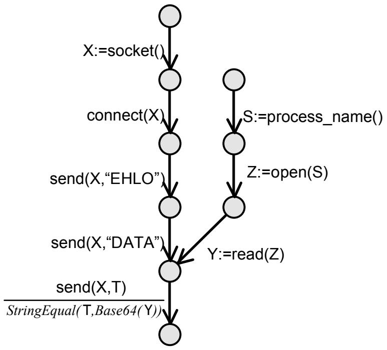
Malspecs Benefits



- ✓ Symbolic variables
- ✓ Constraint-based execution order
- ✓ Independent of obfuscation artifacts

Expressive to describe even obfuscated behavior.

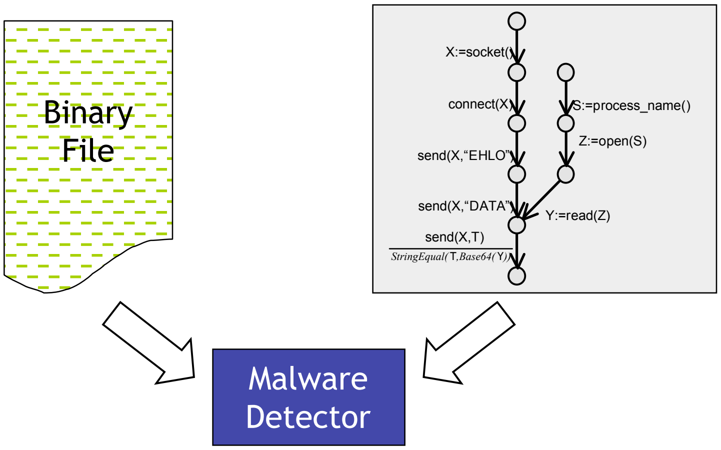
Malspec Detection Strategies



- Static analysis
- Dynamic analysis
- Host-based IDS
- Inline Reference Monitors

Malspecs are independent of detection method.

Detection of Malicious Behavior



Binary File

X:=socket()

connect(X)

send(X,"EHLO")

send(X,"DATA")

send(X,T)

StringEqual(T,Base64(Y))

S:=process_name()

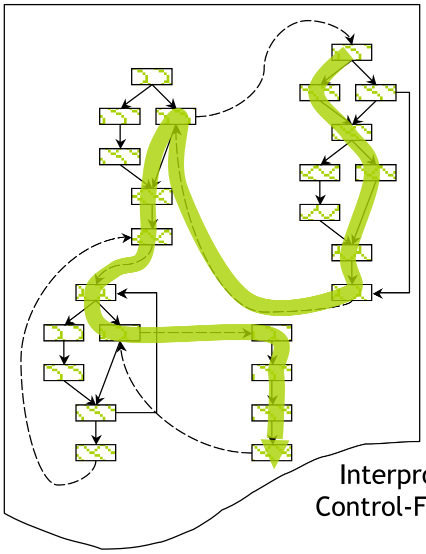
Z:=open(S)

Y:=read(Z)

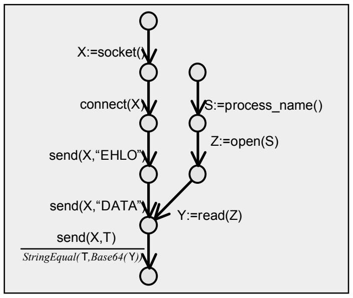
Malware Detector

Goal: Find a program path that matches the malspec.

Find A Malicious Program Path



Interprocedural Control-Flow Graph



X:=socket()

connect(X)

send(X,"EHLO")

send(X,"DATA")

send(X,T)

StringEqual(T,Base64(Y))

S:=process_name()

Z:=open(S)

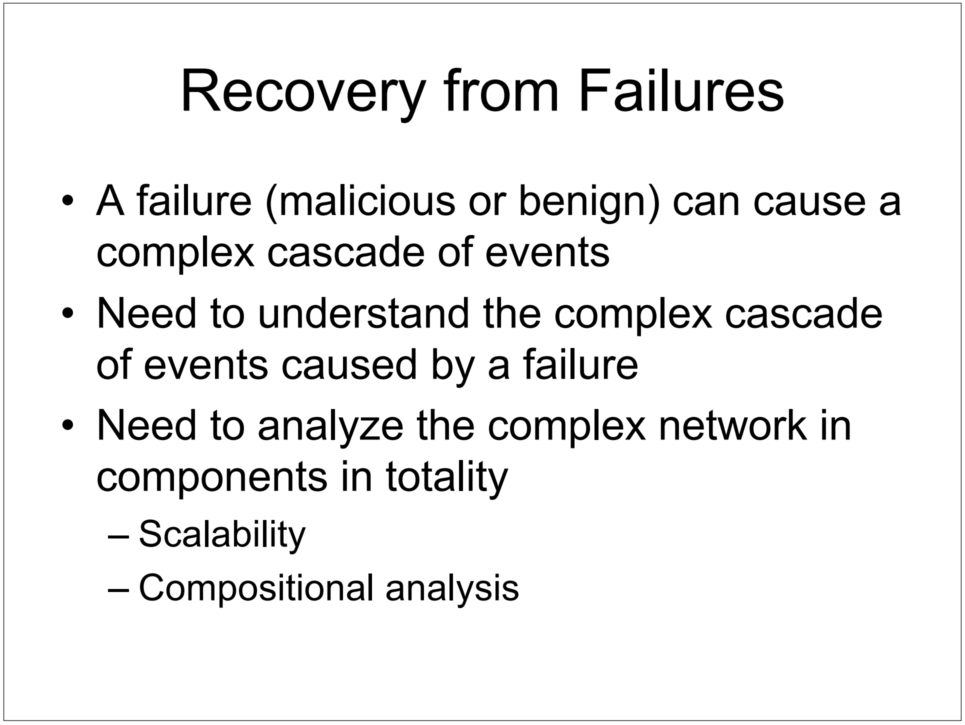
Y:=read(Z)

Stable Environment Assumption

- All the above mentioned work assumes a “nearly” stable environment
- Example: web server
  - Is configurable, but the environment is not that rich
  - Environment is not too dynamic
  - Not rich interaction with other components
- Incorporating “dynamic environments” into the techniques described before is a challenge

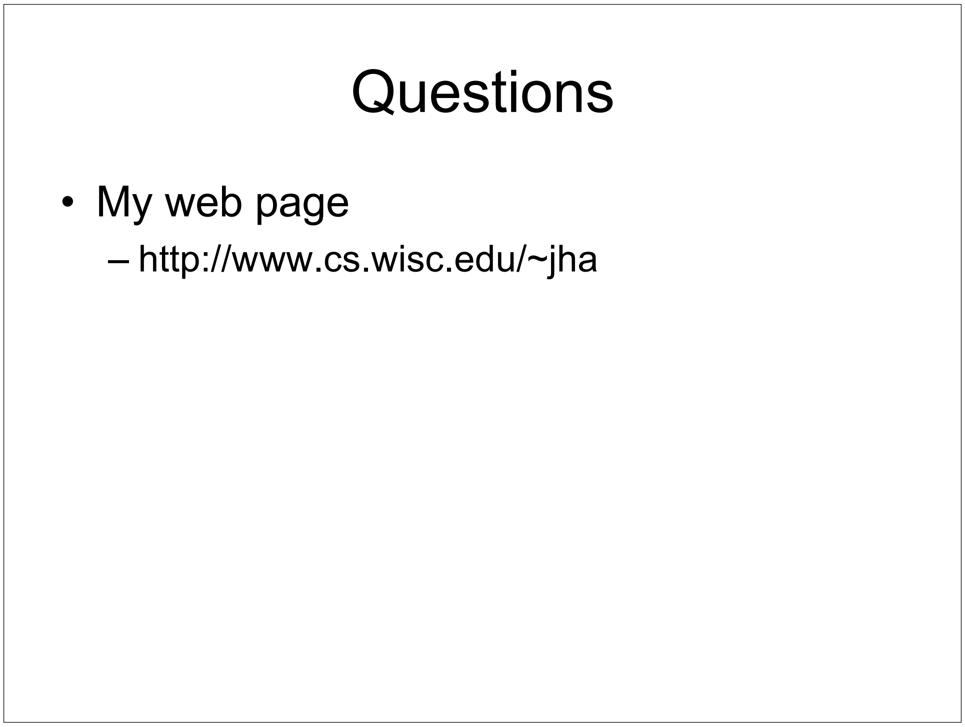
Vulnerability Assessment in Presence of a Dynamic Environment

- Dynamic and static analysis techniques assume a relatively stable environment
- Parameterized static analysis
  - Parameterize static analysis with environment assumptions
  - Similar to assume-guarantee reasoning in model checking
- Parameterized vulnerability signatures



Recovery from Failures

- A failure (malicious or benign) can cause a complex cascade of events
- Need to understand the complex cascade of events caused by a failure
- Need to analyze the complex network in components in totality
  - Scalability
  - Compositional analysis



Questions

- My web page
  - http://www.cs.wisc.edu/~jha

New Direction for Software Protection in Embedded Systems

Kang G. Shin

Department of EECS
University of Michigan

Feb 22, 2007

Background

- Why application software protection in distributed embedded systems?
  - In embedded systems, application programs perform mission-critical tasks and carry sensitive info
  - Privacy/integrity of these applications is critical to the security and robustness of any distributed embedded system
- Current approach: have the OS protect the applications
  - E.g., OS provides process isolation, crypto services, etc.
  - Apps must trust OS. OS should be protected by hardware
  - Processor protects OS via user/kernel separation

My Position

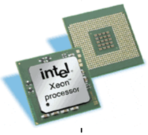
Classical user/kernel separation is too coarse to confidently protect app software with high security needs. The limitation should be overcome by creating a new protection system.

Classical User/Kernel Separation

Kernel mode

Processor state with privilege

- Execution mode for OS
- Ability to execute all instructions
- Unrestricted access to hardware



User mode

Processor state without privilege

- Execution mode for applications
- Can't execute system instructions
- Restricted access to hardware

- An autocratic model for separation of power: the kernel code executes with absolute power
- Entire security of the system hinges on the trustworthiness of the kernel mode software (i.e., OS)
- Effective for protecting OS, but *this simple dichotomy is too coarse and there are several limitations*

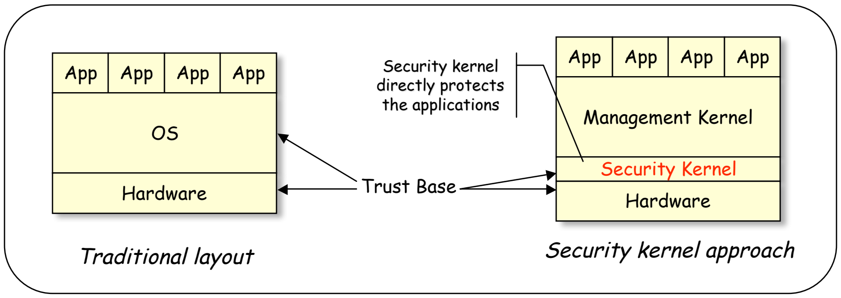
Limitations of user/kernel Dichotomy

1. No defense against OS compromise
   - There is no effective 2nd line of defense to applications
   - Further protection of applications is meaningless as the attacker can easily disarm any protection mechanism
2. Difficult to reduce the OS verification overhead
   - Trend: OS is becoming larger and is from diverse sources
   - The dichotomy dictates any code that requires even slightest privilege must execute in kernel mode, where the code is subject to complete verification
3. Undue trust dependencies
   - App vendors require OS vendors not to spy on the apps
   - Apps must trust every component of OS
   - Every OS component must be validated (e.g., device drivers)

New Directions for Software Protection

- Need a new protection system
  - Protect applications even in case of OS compromises
  - Lessen the kernel verification overhead
  - Break trust dependencies
- Challenges in designing such a protection system
  - Identifying an appropriate threat model
    - Model that captures essential security needs of apps
  - Preserving OS's management power
    - Restriction by the new protection shouldn't obstruct OS's job
  - Finding a small and simple enforcing mechanism
    - Implementation must be easily verifiable
- My proposal: Separate security from management
  - The new protection system protects privacy/integrity of apps.
  - It is implemented by a 'security kernel' (continue)

Security Kernel vs. Mgmt Kernel

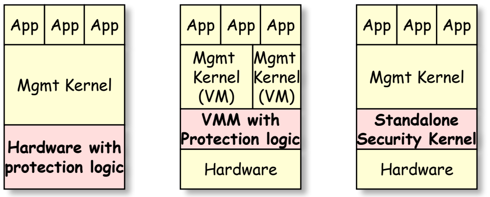


Traditional layout

Security kernel approach

- Traditional OS → Security kernel + Management kernel
- Management kernel is responsible for resource management
- Security kernel is a thin layer enforcing the new protection system
  - It directly protects privacy/integrity of applications data
  - Applications are protected even if the management kernel is compromised
- Management kernel doesn't have to be trusted by applications

Implementation Alternatives



1. Hardware 2. VMM 3. Standalone

	To Verify & secure	To realize	To deploy
Hardware	Easy	Hard	Hard
VMM	Hard	Easy	Easy
Standalone	Easy	Easy	Easy

1. Hardware
   - Processor is modified to implement the protection logic
2. Software: Using virtual machine monitor (VMM)
   - A VMM, sitting between HW and OS, can be utilized
   - Due to size/complexity, verifying the VMM is challenging
3. Software: Standalone security kernel
   - A thin layer implementing only the protection system
   - Can be made small and simple, thus easy to secure

Impact

- Paradigm shift in designing secure distributed embedded systems
  - The new application protection system relaxes many assumptions currently made
  - To ensure the security of application software, management OS no longer has to be trusted
  - It enables implementing ambitious distributed systems which were too risky under the assumption of trusted OS

Conclusion

- Another system of protection is needed to overcome the limitations inherent with the coarse classical user/kernel separation.
- The protection system must
  - Protect applications even in case of OS compromises
  - Lessen the kernel verification overhead
  - Break trust dependencies
- We have been exploring approaches for implementing such a protection system

Session IV – Hardware Security

Session Chair: Tao Xie

Position Paper: *ARO Workshop Security of Embedded Systems and Networks*

Sean Smith
Dartmouth College
http://www.cs.dartmouth.edu/~sws/

February 4, 2007

I'll start with two caveats. The first is a matter of scale. Trying to write this position paper leads me to a conundrum. The workshop's grand overture poses problems and issues that excite me, and spark research ideas. "Hey, maybe we could address problem Z by combining techniques X and Y!" However, that is the stuff of a specific proposal (and might become one, if my colleagues cooperate); the workshop invitation then jars me out of this low-level vision to a high-level one. "What are the three most superlative...."

- **Caveat 1:** It's hard to distinguish the "big picture" from the smaller details.

The second is a matter of tone. Before I came back to academia, I spent time working for the government (advising folks worried about security in real-world deployments) and then industry (building security-motivated embedded systems in the real worlds). But academics often value "intellectual depth" over the real world.

- **Caveat 1:** It's hard to distinguish the "fundamental science" from what's merely an important problem or promising tool for a real-world problem.

Fundamental limitations of today's security mechanisms

Not Thinking about Enough Levels. My generation came of age at a time when the only way for a young teen to get a computer was to build one—but in those early days of 8-bit microprocessors, one actually *could.* History forced a computing education that started at almost the transistor level and worked its way up. This approach had downsides, such as a predilection for C and a parsimony for saving bytes. But it had an upside as well: one starts to see that there are many layers to what we consider as "computation." However, perhaps as a result of the increasing complexity of computing systems, we see too many security mechanisms (for embedded systems—or any other system) focus on only one level.

- Use of typesafe languages may eliminate certain classes of vulnerabilities—but may come at a performance and usability cost (ever try to code in Clay?) and have been successfully compromised via light-bulb-induced memory errors. (Looking only at the language level also neglects how much of the underlying libraries, OS, and firmware may be written in very primitive languages, such as assembler.)
- Hardware techniques such as trusted platform modules and the TCG architecture bind secrets and attestations to a system configuration expressed by a series of hashes. However, mapping from this expression to the properties that relying parties really care about is not obvious (and perhaps not even tractable). What's worse, until the emergence of resettable PCRs, this "configuration" ended up including the whole operating system. Another example of this limitation is the contortions and reworking required to mesh the TPM-based architecture with the full power of virtualization.
- Secure architectures don't guarantee that their applications won't have flawed APIs. (As a member of the 4758 team, I know this one from personal experience.)
- Hardware tokens that protect private keys fail to take into account the integrity and authentication of the software systems that request their use.
- Thinking about computation solely in the logical terms of a box with binary inputs and outputs neglects the fact that the box functions as a physical device in the physical world. This oversight has led to over a decade of side-channel attacks in the open world—and even perhaps to last week's fun of using audio content on Web sites to launch privileged commands on Vista.
- Thinking about a hardware device as some set of modules specified in VHDL or Verilog neglects to take into account that whether the netlists and layout that comes out at the end correctly embody the specification that went in.

- The recent calls for a "Cyber Manhattan project" seem to implicitly assume that the computer security crisis can be solved with technology alone. However, success requires navigating human, policy, and economic angles as well. The original Manhattan project only had to build a few bombs—it didn't have to change the way all of humanity used refrigerators.

One might even go out on a limb and say that we need a new kind of composition theorem: not between "peer" modules, but rather across layers and organizational boundaries.

Cryptography's Questionable Future. Sending a computing system out into the cold, cruel world is scary—particularly for embedded devices, which often must fend for themselves. Security architectures to control code updates and to authenticate devices are based on cryptographic protocols. Best practices tell how to soundly engineer protocols from the primitives and keylengths that cryptographers deem secure. However, the security of these primitives are themselves based on assumptions. History has shown these assumptions don't always remain true as long as anticipated. Here one cite ambitious predictions about RSA and DES; the slow decline of MD5 over ten years; or the lesser-noticed rapid decline of the ISO9796 padding function. The elephant on the table now is SHA1; the TCG meetings just last week were full of heated discussion on how to add "hash agility" to the TPM specification—and the pushback from businesses who don't want to commit product hardware to a specification that is proving malleable.

Keeping Secrets. Sending a self-protecting embedded system out into the cold, cruel world also often requires that the system itself be able to keep and use cryptographic secrets. Here again, history shows that keeping secrets is difficult. In the open world, one need only look at Ross Anderson's lab for a good sampling of how researchers have made repeated hash of low-cost "tamper-protected devices" and exploited amusing side-channels. A few promising techniques have come along—our 4758 device has yet to have its core protections broken (to our knowledge), but is rather expensive and brittle; the SPUF work from MIT looks interesting. However, a method by which inexpensive, environmentally durable devices can keep secrets with high assurance—and a method to have assurance that they can—eludes us (at least in the public world).

Promising innovations and abstractions for future systems

TCG. I could go on for a long time on all the problems with the TCG organization and architecture. The standard quips—such as "designed by committee, and it shows"—suffice. However, to paraphrase someone, the important thing here is not that the bear dances well, but that it dances at all. Fifteen years ago, we could play with secure coprocessor prototypes and say "pretend that all commodity devices had something like this built in." Now they do, sort of. Getting an idea to permeate the broader infrastructure is the hard part, but industry has now done that, and looks poised to continue being amenable to such thinking.

Multicore. Perhaps as a consequence of trying to adhere to Moore's Law, hardware vendors are now giving us more cores than we know what to do with—and are desperately searching for applications, usage models, and a business case. This position gives us an opportunity to rethink what a CPU "does"—and to see these changes happen in the real world.

Calls for a principled revolution. Voices over recent years (e.g., at the CRA Grand Challenges in Infosec workshop a few years back, or at the NSF Safe Computing Workshop last fall, or at the Cyber Trust PI meeting last week) have repeated the observation that things aren't working, that the current approaches to securing computing systems aren't working, and that (to paraphrase Rich DeMillo) we're not "likely to program our way out of this mess." The right questions are being asked.

Possible milestones

This part is harder for me. I might joke that it's because I've already pushed the limits of my space budget; however, the reality is that it's hard for me to describe the end result of a research program that hasn't happened yet.

Secure Processing On-Chip

Hsien-Hsin S. Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250
leehs@ece.gatech.edu

Santosh Pande
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
santosh@cc.gatech.edu

ABSTRACT

Providing security in embedded systems is in urgent needs while there are many challenges in both software and hardware sides that require further research to understand their implications. This paper discusses microarchitectural and compiler support to address a variety of vulnerabilities due to physical tampering, program behavior exploits, and digital rights management issues. We also advocate the need for protecting intellectual properties programmed in the growing number of FPGA-based embedded systems.

1. INTRODUCTION

While embedded computing is becoming more pervasive and invisible, the ways users communicate and operate data on these devices, however, are becoming more vulnerable to malicious exploits. When these data, either sensitive or insensitive, are manipulated in a way they are not intended for, some dire consequence may ensue. For example, crackers can reverse-engineer the cryptographic keys of a multimedia system or game console to duplicate and distribute illegal copies of proprietary software [1]. Another example described in [2] shows that well-resourced crackers can invade one's privacy by monitoring the thermostat to determine if one is at home or not. Even worse, malicious attackers can change the setting of the thermostat through Internet and damage pipes or kill pets during winter times.

To provide reliable security for these devices to combat against various types of attacks remain a major challenge to both hardware and software designers. The reality is that embedded system designers can no longer consider security as an afterthought as many robust security features require shrewd and thorough consideration at the very early design stage. In this paper, we discuss potential security breach from a system's perspective at the microarchitecture level and the compiler level. We hope our advocates will raise the consciousness of building security as an indispensable part in the embedded system design flow.

2. PHYSICAL TAMPERING

One of the greatest concerns on embedded devices is regarding malicious exploits via physical tampering of the devices when adversaries gain full physical access to the hardware and reveal sensitive data or intellectual property (IP) algorithms employed in these compromised devices. Obviously, secrets inside these devices must be protected against these physical attacks. However, the issue becomes even more challenging when both IP protection requirement and real-time constraint need to be met for these embedded applications. To guarantee both criteria, hardware-based encryption support is generally implemented to provide satisfactory performance while caution must be made to not adding too much cost to the systems. Nevertheless, employing encryption alone is not sufficient to avoid new types of attacks via other new breed of attacks such as using side channels [3, 4, 5]. Vulnerability can be exploited by analyzing information leaked through these channels. For example, the absolute and relative locations of the program code are not altered during instruction fetch. In other words, addresses are issued on the bus as plaintext and can be probed by crackers to reconstruct the control-flow graph of a program. Such a vulnerability is particularly pronounced in embedded processors, which typically do not employ cache hierarchies for the requirement of predictable timing. Even with the presence of an instruction cache, a cracker can still easily circumvent the cache by turning off the cache or flushing the cache to force instruction addresses shown on the external bus. In some cases, such information leakage can lead to the revelation of critical information such as encryption keys or passwords of the compromised systems. Another example of the same type of exploits is differential power analysis (or DPA). As shown in previous studies, a well-equipped and motivated cracker can perform non-invasive power (or current) analysis by using oscilloscope on an embedded device such as Smart Cards to retrieve secrets. The idea is based on the observation that power dissipation is strongly correlated to different program behavior on a processor, which can then be used as a signature to compromise secrets. Furthermore, the growing application of low-power techniques such as clock gating makes such attacks even easier.

To combat such issues, effective and efficient obfuscation techniques must be considered, in particular, building them directly into the hardware at the microarchitectural and circuit levels. Fundamentally, obfuscation is aimed to randomize any trace or signature exhibited from address stream or measurable power or current consumption, making distinctive computation operations indistinguishable. A solution demonstrated by [6] uses an on-chip shuffle buffer to perform randomization for the address footprint. The shuffle buffer, essentially an extended small memory array but exclusive to the memory, was designed to reorder all addresses to the memory, obfuscating the address recurrences. Addresses that are ready to be evicted from the shuffle buffer due to a conflict will swap their locations between the shuffle buffer and the main memory. As such, the same address request will appear differently on the bus every time and the goal to evenly distributing the observed addresses can be achieved. Several other literature [7, 8] also investigated such address leakage issues for different system platforms.

3. CONTROL FLOW VULNERABILITY

Exploits such as buffer overruns that alter the program behavior by injecting malicious codes or manipulating high-privileged users inputs represent another major concern. The latter often interacts with input channels such as keyboard or network connection and changes the intended program flow to accomplish their illegitimate actions. Note that a pure software countermeasure can be slow and incapable of detecting such violation. To make the software more robust and evident to such attacks, anomaly detection mechanisms need to be established. An anomaly system is aimed to monitor program execution and raise an alarm whenever there is a detected abnormal program behavior such as program is redirected to unintended or undefined program paths.

An effective mechanism requires to enforce the control-flow awareness via compiler's analysis and microarchitectural support to enable the protection with high efficiency and high accuracy. For instance, an Infeasible Path Detection System (IPDS) proposed in [9] explores the synergy of compiler and microarchitecture to counteract such memory tampering attacks causing invalid program control flow. In the proposed system, the compiler analyzes correlations among conditional branches to realize illegal program flow changes. Then the collected information is made available to the runtime system. The runtime system, with the support of small hardware tables, will detect dynamic violation of infeasible program paths based on the static information.

4. DIGITAL RIGHTS MANAGEMENT

With the emergence of online commerce on virtual properties such as 3D game characters or arts, to protect these intellectual property on embedded devices and to restrict their usage have become a new design challenge. The recent incident of hacking Xbox [1] furthers the urgent need to include native hardware support for providing a more robust digital rights management (DRM) to enable a tamper-proof embedded platform. To integrate such protection scheme into media processing systems more seamlessly and securely without compromising performance, it requires that security experts and embedded hardware and software designers to align their tasks together. A DRM-enabled 3D graphics processor was demonstrated in [10]. It consists of two components, a cryptographic unit that decrypts protected IP data, and a license verification unit that authenticates the license of these data. Similar to digital rights licenses used in other content protection scenarios, the graphics digital rights licenses released by their providers specify and designate the desired usage of the graphics data. Under this system, exploits are prevented by restricting the otherwise arbitrary bindings among geometry input, textures and shaders through the licenses that define the legal bindings of these objects. During rendering, the binding context consisting of decryption keys and digests of protected data will be checked and verified in the cryptographic hardware units. Additionally, such a DRM-enabled graphics system also protects the Z-buffer, i.e. the depth information, to prevent crackers from reconstructing a 3D geometry model by dumping out the Z-buffer values.

5. IMPLICATIONS OF FPGA-BASED DESIGN

More recently, due to the substantial improvement in FPGA technology, digital designs using FPGA is no longer simply for early prototype or proof-of-concept. In fact, products are being implemented using FPGA for its efficiency (design turnaround time), reconfigurability, and flexibility. FPGA is also an attractive solution for implementing cryptographic applications to adapt the needed changes and enhancements in security policies. An example is set-top boxes which use FPGA to encrypt and decrypt the media stream for pay-per-view movies. Even though the above applications seem to fall into two different groups, yet their demands in security are almost identical — i.e., how to protect the contents implemented and configured in the FPGA? The contents from the first category are related to the IP (i.e. the algorithms) issues of a proprietary design, while the contents from the second category will contain critical secrets such as the cryptographic keys. Similar to what we described earlier, FPGA-based designs suffer from physical tampering — from IP theft by simply reading bitstream out of the FPGA to DPA side-channel attacks. To address such vulnerabilities, new ideas are needed for both FPGA chip vendors and synthesis tools and algorithms to protect the contents programmed on the gate arrays.

6. CONCLUSION

We are entering an interesting time for embedded designers to (re)consider security as a top design priority at the early design stage. The problem is multi-faceted, involving all layers in a design including the system software (OS and compiler), architecture, microarchitecture, and circuits. Several challenges are lying ahead and a holistic solution across the stack is in need.

In this paper, we are advocating to integrate inherently high security hardware and system support to embedded processors. These schemes typically require dedication of on-chip hardware resources being used to achieve high efficiency and be effective. Nevertheless, any additional hardware feature for cost-constrained embedded systems must be carefully evaluated and justified. Another challenge of integrating security solutions in embedded systems is power consumption, which is already a constraint for battery-powered devices. It will become worse when obfuscation techniques are applied to randomize and disguise program behavior. Adding security to both compiler and hardware levels could also procrastinate the design turnaround time, a critical cost and competitiveness concerns given the short time-to-market cycles of these products. All these trade-offs need to be deliberately balanced in the design of future embedded systems to enable highly security processing.

7. REFERENCES

[1] Andrew Huang. *Hacking the Xbox: An Introduction to Reverse Engineering*. No Starch Press, 2003.

[2] Philip Koopman. Embedded System Security. *IEEE Computer*, pages 95 – 97, July 2004.

[3] P.C. Kocher, J. Jaffe, and Jun B. Differential Power Analysis. In *Proceedings of Advances in Cryptology, Crypto 1999*, 1999.

[4] Weidong Shi, Hsien-Hsin S. Lee, Chenghuai Lu, and Mrinmoy Ghosh. Towards the Issues in Architectural Support for Protection of Software Execution. *SIGARCH Computer Architecture News*, 33(1):6–15, 2005.

[5] Weidong Shi and Hsien-Hsin S. Lee. Authentication Control Point and its Implications for Secure Processor Design. In *Proceedings of the 39th Annual International Symposium on Microarchitecture*, pages 103–112, 2006.

[6] Xiaotong Zhuang, Tao Zhang, Hsien-Hsin S. Lee, and Santosh Pande. Hardware Assisted Control Flow Obfuscation for Embedded Processors. In *Proceedings of the 2004 International Conference on Compilers, Architectures, Synthesis on Embedded Systems*, pages 292–302, Washington D.C., 2004.

[7] Lan Gao, Jun Yang, Marek Crobak, Youtao Zhang, San Nguyen, and Hsien-Hsin S. Lee. A Low-Cost Memory Remapping Scheme for Address Bus Protection. In *In Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, pages 74–83, September 2006.

[8] Xiaotong Zhuang, Tao Zhang, and Santosh Pande. HIDE: An Infrastructure for Efficiently Protecting Information Leakage on the Address Bus. In *the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 72–84, 2004.

[9] Xiaotong Zhuang, Tao Zhang, and Santosh Pande. A Low-Cost Memory Remapping Scheme for Address Bus Protection. In *In Proceedings of the 39th International Symposium on Microarchitecture*, pages 113–122, September 2006.

[10] Weidong Shi, Hsien-Hsin S. Lee, Richard M. Yoo, and Alexandra Boldyreva. A Digital Rights Enabled Graphics Processing System. In *In Proceedings of the ACM SIGGRAPH/Eurographics Workshop of Graphics Hardware*, pages 17–26, 2006.

A Case for Tamper-Resistant and Tamper-Evident Computer Systems

Yan Solihin
Center of Efficient, Secure, and Reliable Computing (CESR)
North Carolina State University
solihin@ncsu.edu

Abstract

Recent industrial efforts in architectural and system support for trusted computing still leave systems wide-open even to relatively simple and inexpensive hardware-based attacks. These attacks attempt to snoop or modify data transfer between various chips in a computer system such as between the processor and memory, and between processors in a multiprocessor interconnect network. Software security protection is completely exposed to these attacks because such transfer is managed by hardware without any cyptographic protection. In this paper, we argue that the threats from such attacks are serious and urgent, and that computer design should place a priority in protection against these attacks.

1 Fundamental limitations of today's security mechanisms

While data transfer between several computer systems that are networked is managed by software, data transfer within a computer system between its components is managed completely by hardware and is transparent to the software. For each computation task, lage amounts of data are transferred between various chips such as the processor and memory, or between processors in a multiprocessor system. Currently, such data transfer is completely unprotected, which can be snooped or altered through relatively simple hardware devices attached to various buses and the interconnects. This presents a serious security challenge in that even the most secure software protection can be broken because its sensitive information is stored as program variables off the processor chip. Furthermore, by snooping data brought into the processor chip, attackers can reverse engineer code, snoop unencrypted data, or even alter data before it enters the processor chip. Recognizing some of these challenges, industrial efforts have resulted in *Trusted Computing* efforts [9, 15]. Unfortunately, Trusted Computing only addresses a small subset of these attacks. While authentication of certain system software is provided with trusted computing, data transfer is still unprotected against snooping and tampering.

Granted, such hardware attacks require the attackers to have physical access to the computer systems, so they are not commonplace yet. However, we believe that there are several important use scenarios of computer systems in which the possibility for such attacks is quite high and needs to be taken very seriously.

The first scenario is when *attackers has almost unlimited physical access to the system* because they either own it, or they administer it. One example from this scenario is consumer electronics such as game consoles and portable media players. Such systems often come with copyright protection mechanism. Users or owners of the system can repeteadly attack the system in order to break such protection mechanism with a strong financial incentive because such devices are common and the cost of designing the attacks can be amortized over many instances. This seriousness of such attacks has been demonstrated by the commercial success of mod-chips, enabled by unencrypted transfer between the BIOS and the processor chip [4].

Another example of such scenario involves *voting machines*. Since these machines are placed in a great number of sites, it is hard to provide them with complete physical security. It is hard to ensure that administrators of the machines will not tamper the machines, or will not unintentionally let others to tamper with them.

Another scenario is when *attackers has limited physical access to the system but there are non-intrusive and traceless ways to attack the system*. Large multiprocessor systems used for utility or on-demand computing servers are particularly vulnerable. In the utility computing model, companies "lease" resources of a large-scale, powerful servers (e.g. the HP Superdome [10]) to customers who need such resources on a temporary basis or who want to offload their IT operations. These large-scale systems are not under the control of the customers who are using their resources. The customers are likely to be wary about adopting the utility computing model unless the secrecy and integrity of their data can be ensured. In fact, concerns about data privacy have been reported to slow down the adoption of utility computing model [1]. If the server system itself does not ensure data confidentiality and integrity, malicious employees or other attackers who can get through the physical security protecting the machine could easily steal or modify important data. The risk of security attacks by selected employees or parties that have physical access to the machine should not be underestimated. For example, in the case of ATMs, Global ATM Security Alliance (GASA) reported that more than 80% of computer-based bank-related frauds involve employees [6]. In the case of DSM systems used for utility computing, the large amounts of sensitive data in these systems create a financial incentive for the attackers to perform corporate espionage or other malicious intents. To make matters worse, such attacks could be performed without disrupting the system, for example by attaching a simple device to an interconnect wire. Such attacks also do not produce traces that can alert other users about the existence of the attacks. These concerns may prompt customers to demand that DSM utility computing systems be equipped with hardware support for data confidentiality before they would be willing to use those systems. This also suggests that data security in DSM systems will become an increasingly important issue in the future.

2 Important research challenges

One main research challenge is how to *efficiently* ensure *privacy*, *tamper-resistant* and *tamper-evident* properties for a computer system. Privacy requires data transfer to be encrypted so that attackers cannot gain much insight into the data from snooping it. Tamper-resistance requires that data transfer is enrcypted in such a way that it is hard for the attackers to tamper the data in a meaningful way. Finally, tamper-evidence requires authentication of data transfer to detect attack attempts and secure logging to record information of the attacks.

Data transfer between chips must be provided with very low latencies, and any delay due to cryptographic operation can significantly slow down the computer systems. For example, current memory access latency is in the order of 200ns, while decryption operation applied to incoming cache block can easily add 30-50% to the latency. Another important challenge is the space overhead due to storing hash codes. In recent studies, to prevent tampering of data transfer, a Merkle tree of hash codes requires a space overhead of 25%. This is clearly unacceptable in a system where performance or cost are critical issues.

Another main research challenge is how to retain the operability of such system. Since the entire memory is encrypted, secure mechanisms are needed in order for the system to communicate with external devices, such the I/O subsystem.

Another major research challenge is how to securely boot the system. For uniprocessor system, this is relatively simple to achieve, but for multiple processors communicating with each other, we need a mechanism to establish trust between the communicating processes. Traditional protocol such as Kerberos is hard to apply because it assumes the existence of secure software. Secure hardware booting cannot assume that the security software is already running.

3 Promising innovations and abstractions for future systems

A body of research exists on memory encryption and authentication schemes for uniprocessor systems [2, 3, 5, 7, 8, 12, 13, 14, 16, 17]. The main assumption in memory encryption and authentication work is that on-chip data is secure and cannot be observed by attackers, while data that resides anywhere off-chip can be observed and altered by attackers using hardware attacks. Therefore, the goal of memory encryption and authentication schemes is to encrypt and hash data before it leaves the processor chip, and then to decrypt and authenticate it when it is brought back on-chip. Several studies use a direct encryption approach where a block cipher such as AES is used to directly encrypt and decrypt data [3, 7, 8]. However, these approaches add the long latency of the block cipher to the critical path latency of off-chip data fetches. To hide this latency, several studies have examined counter-mode encryption where a data block is encrypted or decrypted through an XOR with a pad [12, 14, 16, 17]. The pad is constructed by encrypting a *seed*, which is typically composed of a per-block counter and the block's address. The security of counter-mode encryption relies on uniqueness of pads, which is maintained by by incrementing the block's counter each time the data is updated. Counter-mode hides decryption latency by caching [14, 16, 17] or predicting [12] the block's counter, so pad generation can proceed in parallel with the fetch of the block's data from DRAM. For authentication, Merkle hash trees have been proposed to protect the integrity of data in memory from data tampering and replay attacks. In the Merkle tree scheme, a tree of Message Authentication Codes is formed over the blocks of data in memory, with the root of this tree always kept on-chip. Data integrity can be verified by computing MACs up the tree to the secure root.

Our own research has advanced the state of the art of counter-mode memory encryption and authentication by enabling the processor to hide cryptographic operation latency so that no noticeable slowdown is observed, for both uniprocessor system [16], and large multiprocessor server system [11].

All such technologies serve as a proof-of-concept that efficient memory encryption and authentication can be achieved. However, many research challenges, such as communication mechanism with the external world, secure booting, and tolerating space overheads, remain unaddressed.

4 Possible milestones for the next 5 to 10 years

Milestones should include a working prototype of secure chips. A prototype requires addressing problems that may not be obvious at the research stage, such as the impact of the design on the Operating System and application software. It is also useful to subject the prototype to various attacks on data transfer to make sure that the protection is reasonably secure and securely implemented. Finally, prototyping requires the changes to existing systems to be reduced to a minimum while still providing strong security.

References

[1] D. Bartholomew. On Demand Computing – IT On Tap? *http://www.industryweek.com/ReadArticle.aspx?ArticleID=10303 &SectionID=4*, June 2005.

[2] B. Gassend, G. Suh, D. Clarke, M. Dijk, and S. Devadas. Caches and Hash Trees for Efficient Memory Integrity Verification. In *Proc of the 9th Intl. Symp. on High Performance Computer Architecture (HPCA-9)*, 2003.

[3] T. Gilmont, J.-D. Legat, and J.-J. Quisquater. Enhancing the Security in the Memory Management Unit. In *Proc. of the 25th EuroMicro Conf.*, 1999.

[4] http://www.modchip.com, 2005.

[5] IBM. IBM Extends Enhanced Data Security to Consumer Electronics Products. *http://domino.research.ibm.com/comm/pr.nsf/pages/ news.20060410_security.html*, April 2006.

[6] M. Lee. Global ATM Security Alliance focuses on insider fraud. *ATMMarketplace, http://www.atmmarketplace.com/article.php? id=7154*, 2006.

[7] D. Lie, J. Mitchell, C. Thekkath, and M. Horowitz. Specifying and Verifying Hardware for Tamper-Resistant Software. In *IEEE Symp. on Security and Privacy*, 2003.

[8] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. MItchell, and M. Horowitz. Architectural Support for Copy and Tamper Resistant Software. In *Proc. of the 9th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2000.

[9] Microsoft Corporation. Microsoft Next-Generation Secure Computing Base – Technical FAQ. http://www.microsoft.com/technet/archive/security/news/ngscb.mspx, 2003.

[10] T. Olavsrud. HP Issues Battle Cry in High-End Unix Server Market. *ServerWatch, http://www.serverwatch.com/news/article.php/ 1399451*, 2000.

[11] B. Rogers, Y. Solihin, and M. Prvulovic. Efficient data protection for distributed shared memory multiprocessors. In *Intl. Conf. on Parallel Architectures and Compilation Techniques*, 2006.

[12] W. Shi, H.-H. Lee, M. Ghosh, C. Lu, and A. Boldyreva. High Efficiency Counter Mode Security Architecture via Prediction and Precomputation. In *32nd Intl. Symp. on Computer Architecture*, 2005.

[13] W. Shi, H.-H. Lee, C. Lu, and M. Ghosh. Towards the Issues in Architectural Support for Protection of Software Execution. In *Workshop on Architectureal Support for Security and Anti-virus*, pages 1–10, 2004.

[14] G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. Efficient Memory Integrity Verification and Encryption for Secure Processor. In *Proc. of the 36th Intl. Symp. on Microarchitecture*, 2003.

[15] Trusted Computing Group. https://www.trustedcomputinggroup.org, 2005.

[16] C. Yan, B. Rogers, D. Englender, Y. Solihin, and M. Prvulovic. Improving cost, performance, and security of memory encryption and authentication. In *Proc. of the Intl. Symp. on Computer Architecture*, 2006.

[17] J. Yang, Y. Zhang, and L. Gao. Fast Secure Processor for Inhibiting Software Piracy and Tampering. In *Proc. of the 36th Intl. Symp. on Microarchitecture*, 2003.

ARO Planning Workshop Security of Embedded Systems and Networks

Hardware-Based Security: Trouble and Hope

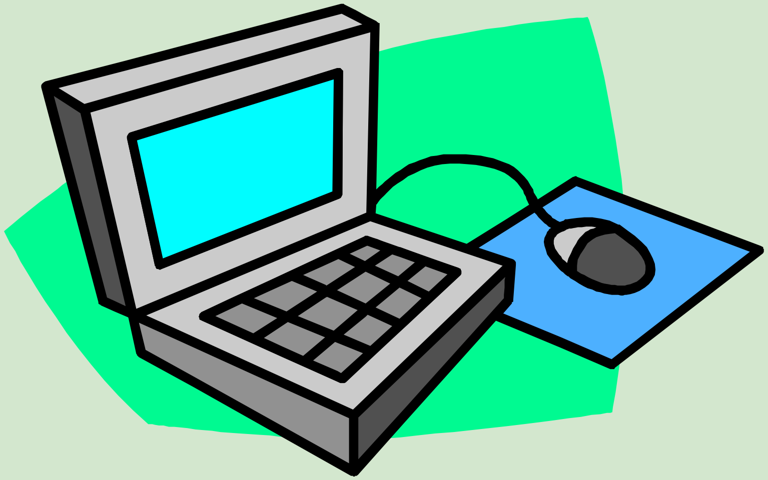
Sean W. Smith
Department of Computer Science
Dartmouth College
Hanover, NH USA

http://www.cs.dartmouth.edu/~sws/

February 22, 2007

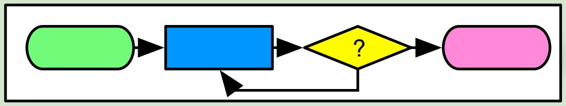
Trouble Area #1:

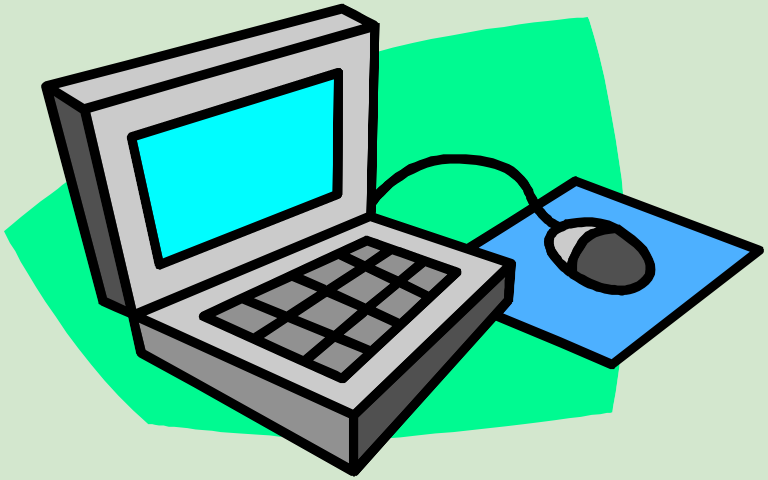
Not Thinking about Enough Levels



Trouble Area #1:

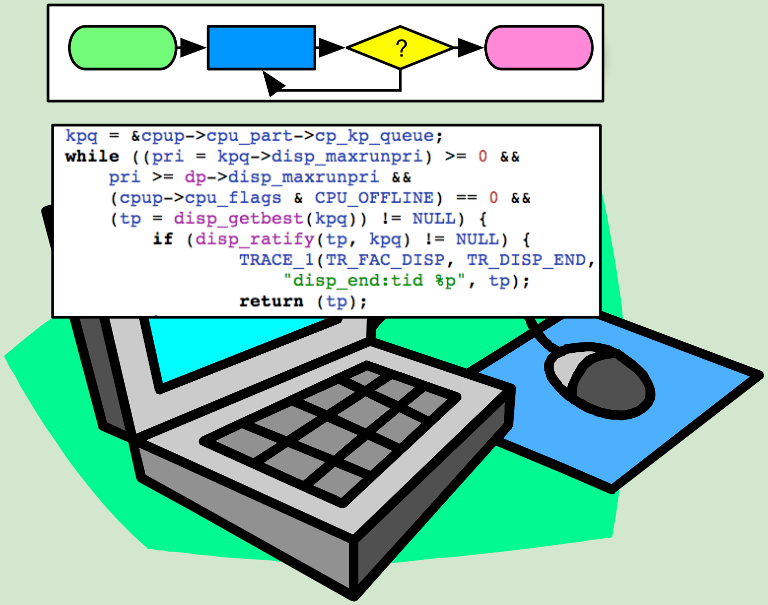
Not Thinking about Enough Levels





Trouble Area #1:

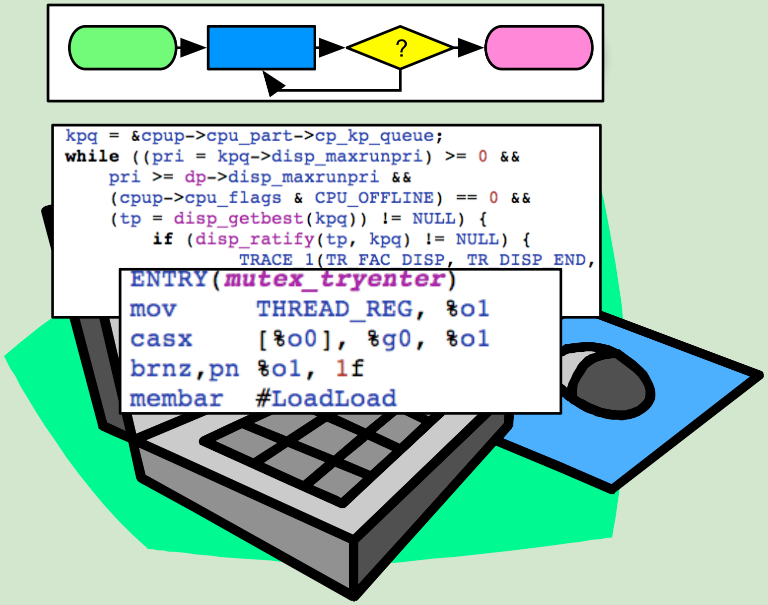
Not Thinking about Enough Levels



```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
                    "disp_end:tid %p", tp);
                return (tp);
```

Trouble Area #1:

Not Thinking about Enough Levels

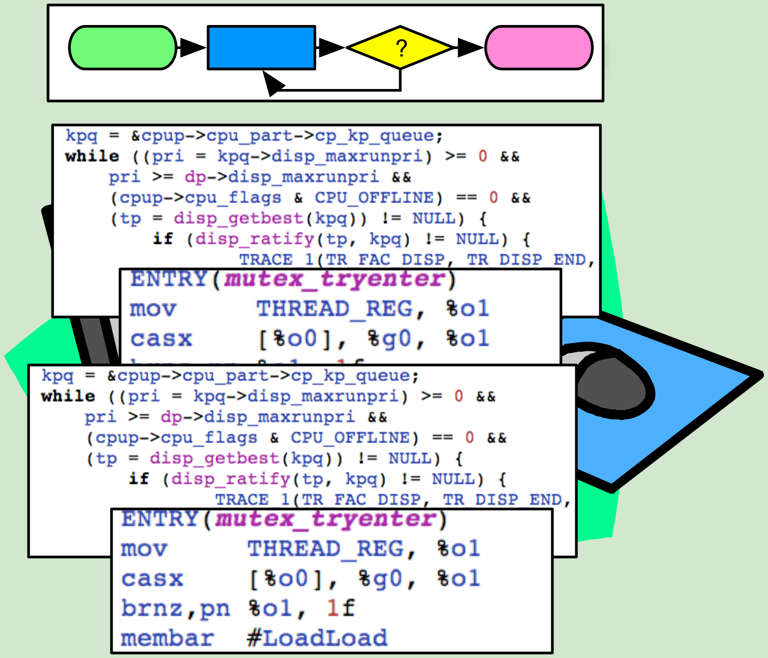


```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov       THREAD_REG, %o1
casx      [%o0], %g0, %o1
brnz,pn %o1, 1f
membar  #LoadLoad
```

Trouble Area #1:

Not Thinking about Enough Levels



```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

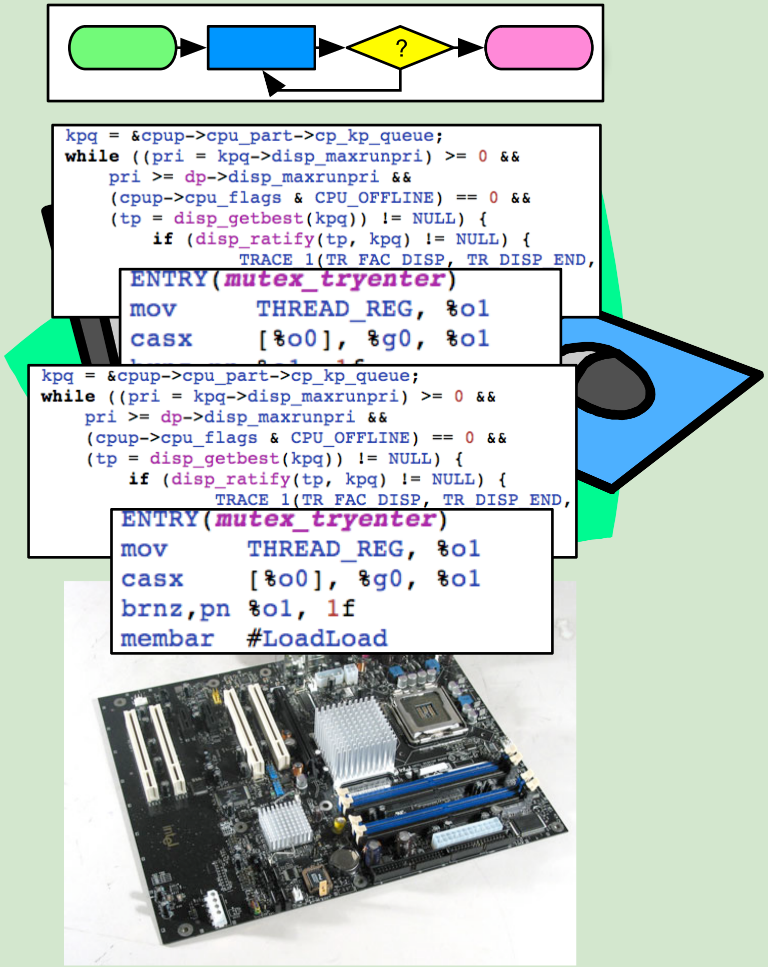
```
ENTRY(mutex_tryenter)
mov       THREAD_REG, %o1
casx      [%o0], %g0, %o1
```

```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov       THREAD_REG, %o1
casx      [%o0], %g0, %o1
brnz,pn %o1, 1f
membar  #LoadLoad
```

Trouble Area #1:

Not Thinking about Enough Levels



```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

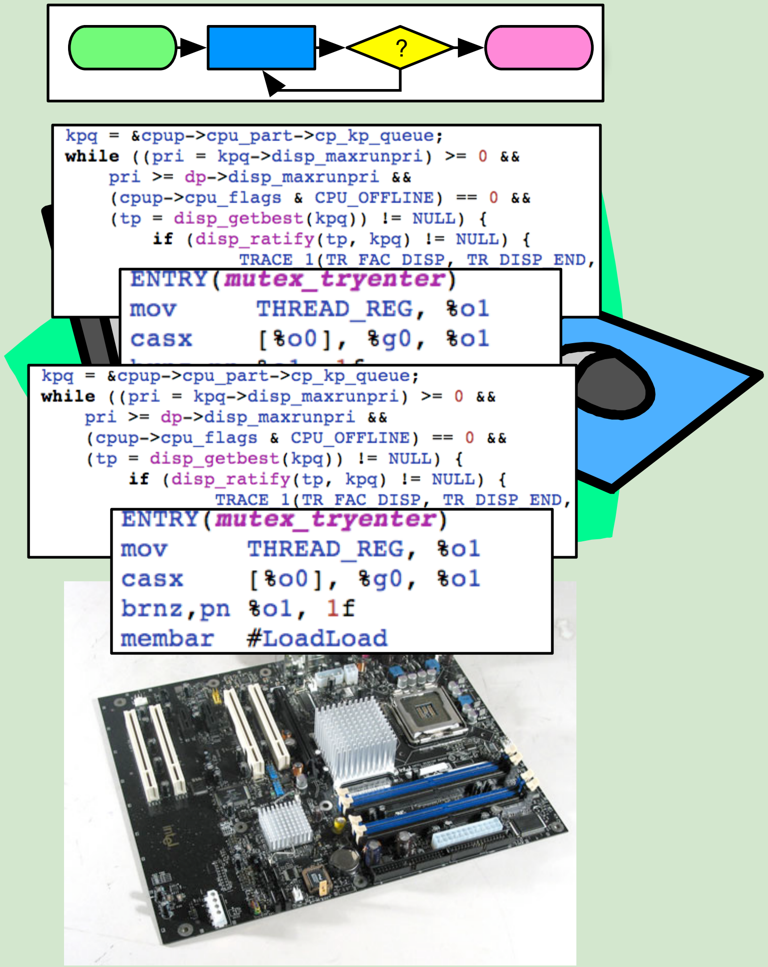
```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
```

```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
brnz,pn  %o1, 1f
membar   #LoadLoad
```

Trouble Area #1:

Not Thinking about Enough Levels



```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

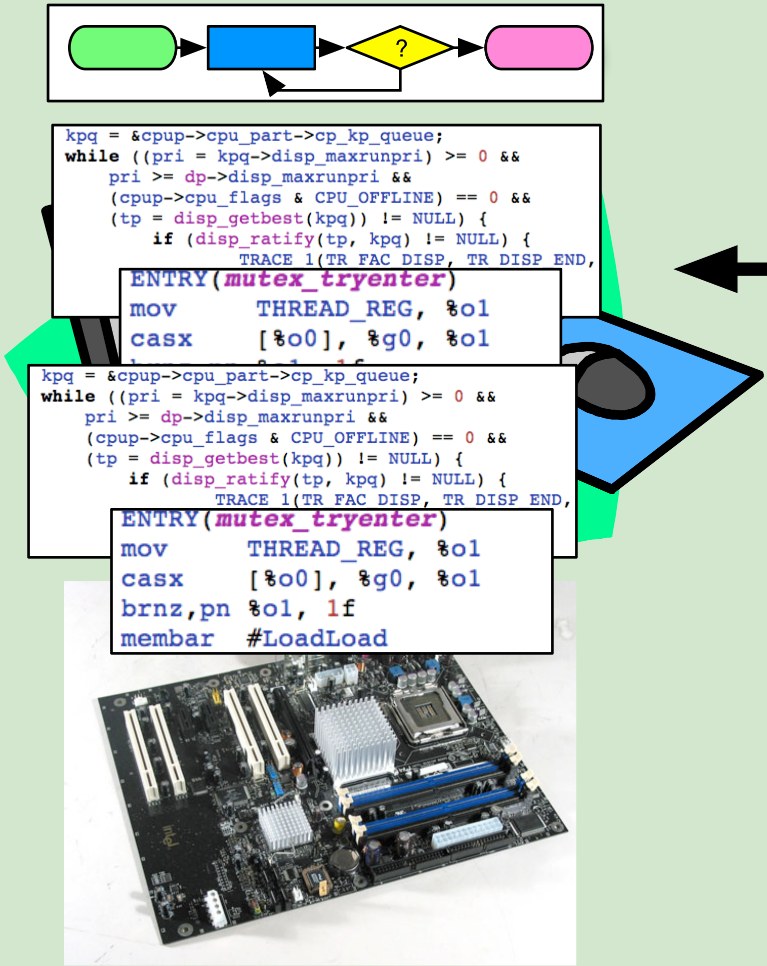
```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
```

```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov      THREAD_REG, %o1
casx     [%o0], %g0, %o1
brnz,pn %o1, 1f
membar  #LoadLoad
```

Trouble Area #1:

Not Thinking about Enough Levels



```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov     THREAD_REG, %o1
casx    [%o0], %g0, %o1
```

```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

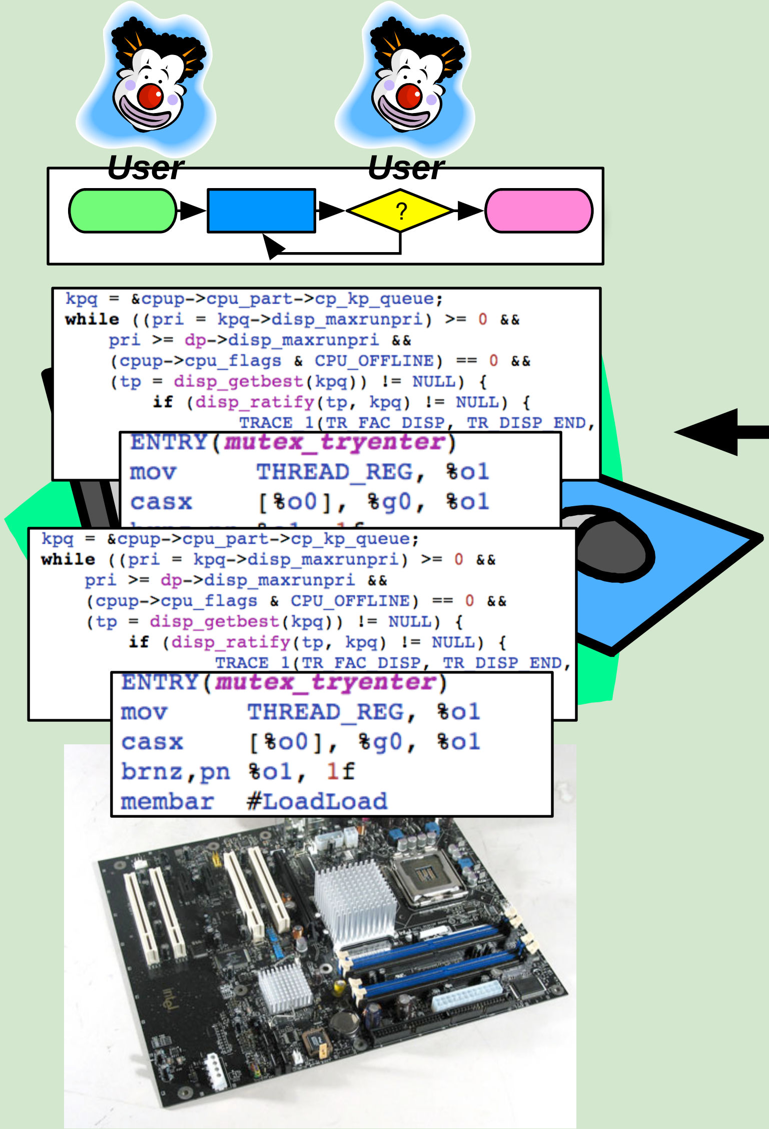
```
ENTRY(mutex_tryenter)
mov     THREAD_REG, %o1
casx    [%o0], %g0, %o1
brnz,pn %o1, 1f
membar  #LoadLoad
```

Trouble Area #1:

Not Thinking about Enough Levels

User **User** **User** **User** **User** **User**

↔



```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov     THREAD_REG, %o1
casx    [%o0], %g0, %o1
```

```
kpq = &cpup->cpu_part->cp_kp_queue;
while ((pri = kpq->disp_maxrunpri) >= 0 &&
    pri >= dp->disp_maxrunpri &&
    (cpup->cpu_flags & CPU_OFFLINE) == 0 &&
    (tp = disp_getbest(kpq)) != NULL) {
        if (disp_ratify(tp, kpq) != NULL) {
                TRACE_1(TR_FAC_DISP, TR_DISP_END,
```

```
ENTRY(mutex_tryenter)
mov     THREAD_REG, %o1
casx    [%o0], %g0, %o1
brnz,pn %o1, 1f
membar  #LoadLoad
```

↔

Examples

- Java type-safety vs. light bulbs

Examples

- Java type-safety vs. light bulbs

- Type-safe C variant for kernel coding vs. kernel coders

Examples

- Java type-safety vs. light bulbs

- Type-safe C variant for kernel coding vs. kernel coders
- hardware-based attestation vs. the computational entity
- hardware-based attestation vs. the operating system

Examples

- Java type-safety vs. light bulbs

- Type-safe C variant for kernel coding vs. kernel coders
- hardware-based attestation vs. the computational entity
- hardware-based attestation vs. the operating system

- IBM 4758 platform vs. API flaws in the CCA app

Examples

- Java type-safety vs. light bulbs

- Type-safe C variant for kernel coding vs. kernel coders
- hardware-based attestation vs. the computational entity
- hardware-based attestation vs. the operating system

- IBM 4758 platform vs. API flaws in the CCA app
- Cyber-Manhattan project vs. economic roll-out

More Trouble Areas

2. Cryptography's questionable future

More Trouble Areas

2. Cryptography's questionable future

More Trouble Areas

2. Cryptography's questionable future

3. Keeping and using secrets

- *effectively*
- *affordably*

More Trouble Areas

2. Cryptography's questionable future

3. Keeping and using secrets

- *effectively*
- *affordably*

Reasons for Hope

1. Industry is open to designing ***and deploying*** hardware-based techniques to enhance security.

TRUSTED
COMPUTING GROUP™

Reasons for Hope

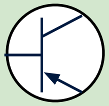
1. Industry is open to designing ***and deploying*** hardware-based techniques to enhance security.

TRUSTED COMPUTING GROUP™

Reasons for Hope

1. Industry is open to designing ***and deploying*** hardware-based techniques to enhance security.

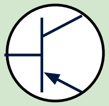
TRUSTED COMPUTING GROUP™

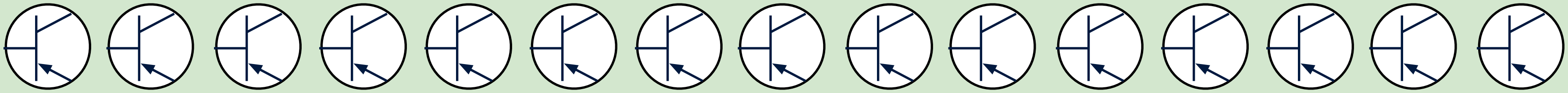
2. The consequences of ***keeping up*** with Moore's Law 

Reasons for Hope

1. Industry is open to designing ***and deploying*** hardware-based techniques to enhance security.

TRUSTED COMPUTING GROUP™

2. The consequences of ***keeping up*** with Moore's Law 

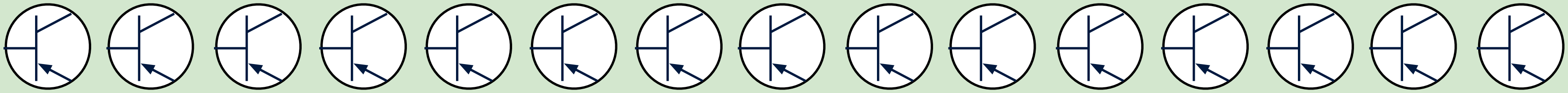


Reasons for Hope

1. Industry is open to designing ***and deploying*** hardware-based techniques to enhance security.

TRUSTED COMPUTING GROUP™

2. The consequences of ***keeping up*** with Moore's Law



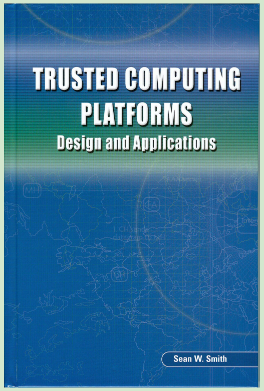
3. Repeated calls for ***principled revolution***

- CRA, I3P, "Cyber-Manhattan Project,"....
- ***This workshop***

Thanks

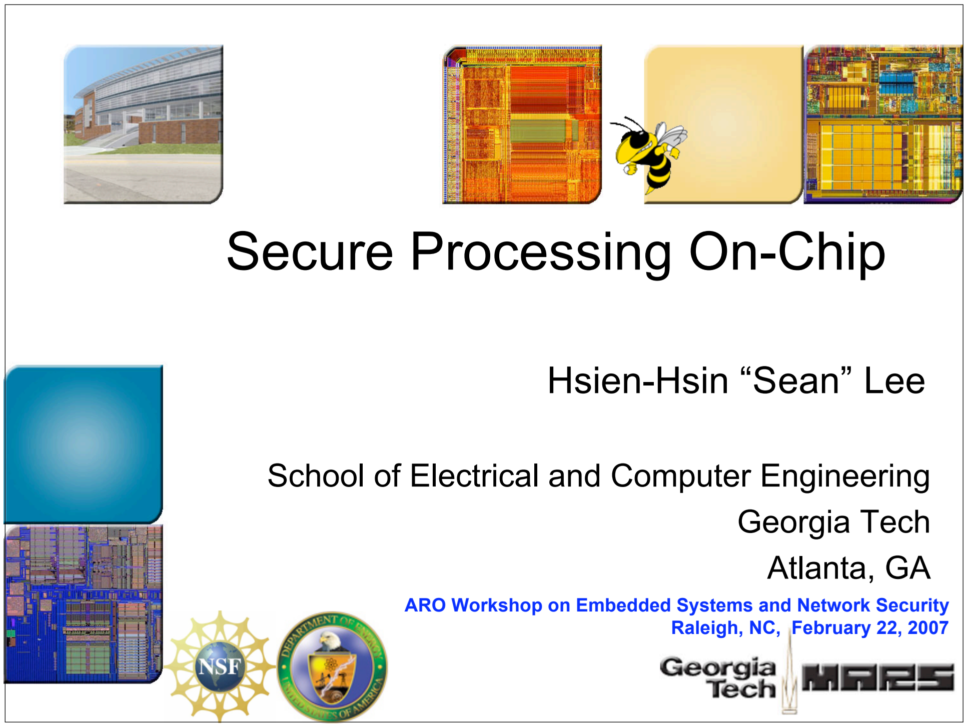
Sponsors:

- NSF CAREER, DoJ/DHS, Mellon, Internet2/AT&T
- Sun, Intel, Cisco (and IBM Research)



For more information:

- http://www.cs.dartmouth.edu/~sws/
- *Trusted Computing Platforms: Design and Applications*. Springer, 2005.

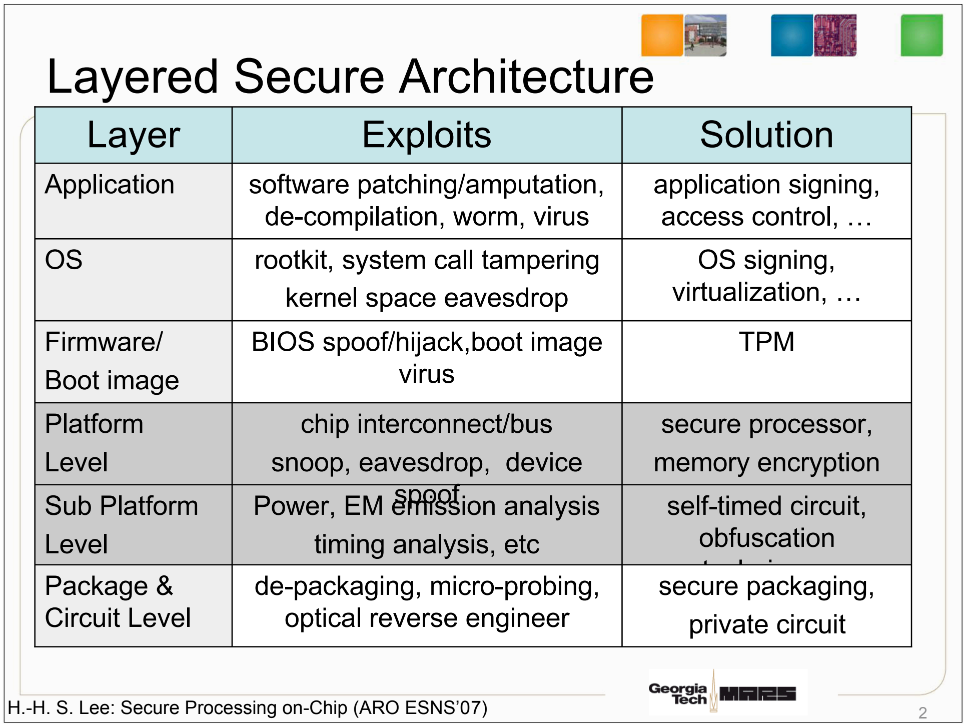


Secure Processing On-Chip

Hsien-Hsin "Sean" Lee

School of Electrical and Computer Engineering
Georgia Tech
Atlanta, GA

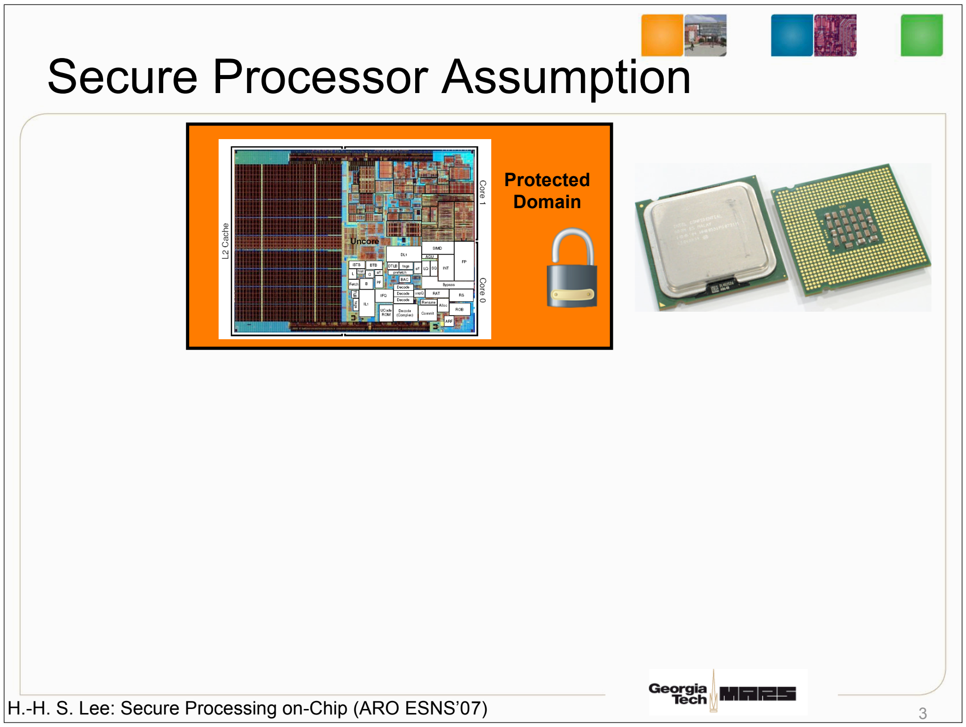
ARO Workshop on Embedded Systems and Network Security
Raleigh, NC, February 22, 2007



Layered Secure Architecture

Layer	Exploits	Solution
Application	software patching/amputation, de-compilation, worm, virus	application signing, access control, …
OS	rootkit, system call tampering kernel space eavesdrop	OS signing, virtualization, …
Firmware/ Boot image	BIOS spoof/hijack,boot image virus	TPM
Platform Level	chip interconnect/bus snoop, eavesdrop, device spoof	secure processor, memory encryption
Sub Platform Level	Power, EM emission analysis timing analysis, etc	self-timed circuit, obfuscation
Package & Circuit Level	de-packaging, micro-probing, optical reverse engineer	secure packaging, private circuit

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07) 2

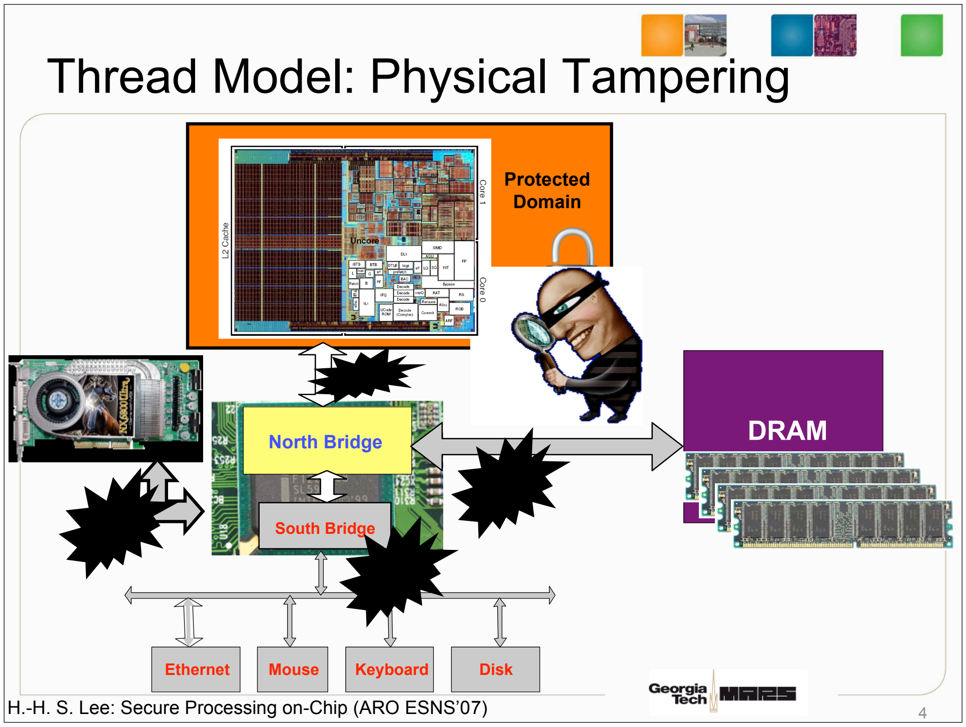


Secure Processor Assumption

Protected Domain

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

3



Thread Model: Physical Tampering

Protected Domain

North Bridge

South Bridge

DRAM

Ethernet

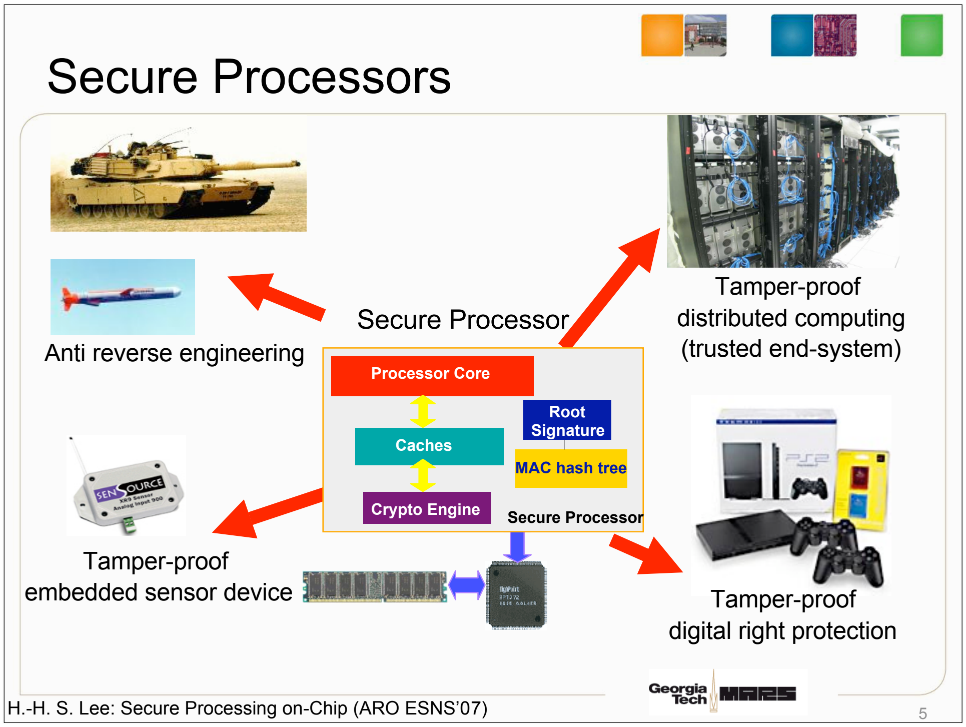
Mouse

Keyboard

Disk

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

4



Secure Processors

Anti reverse engineering

Secure Processor

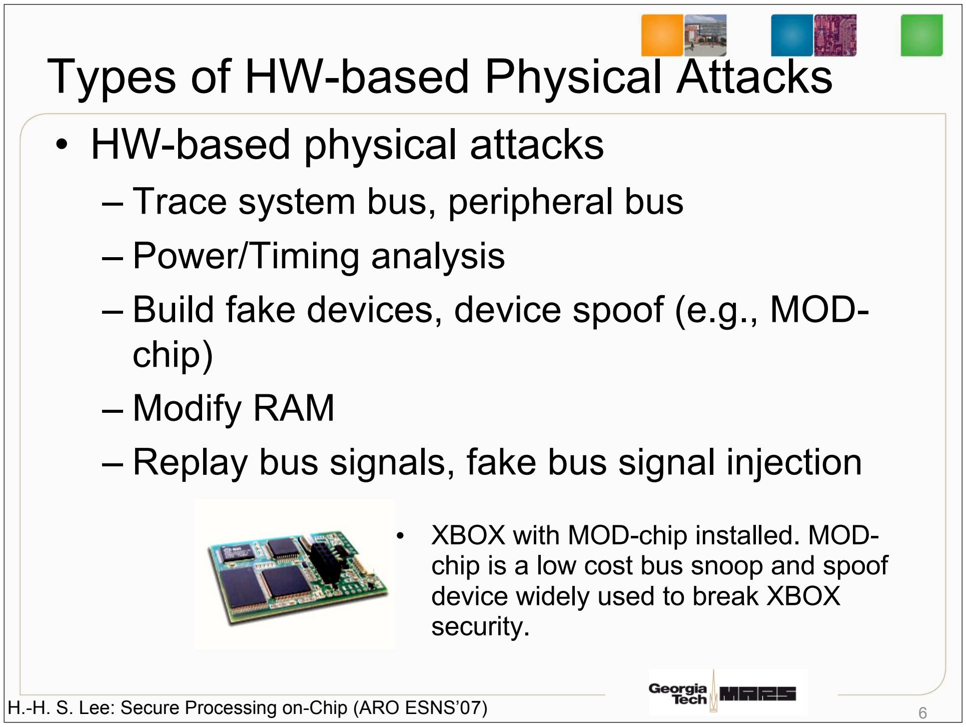
Tamper-proof distributed computing (trusted end-system)

Tamper-proof embedded sensor device

Tamper-proof digital right protection

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

5



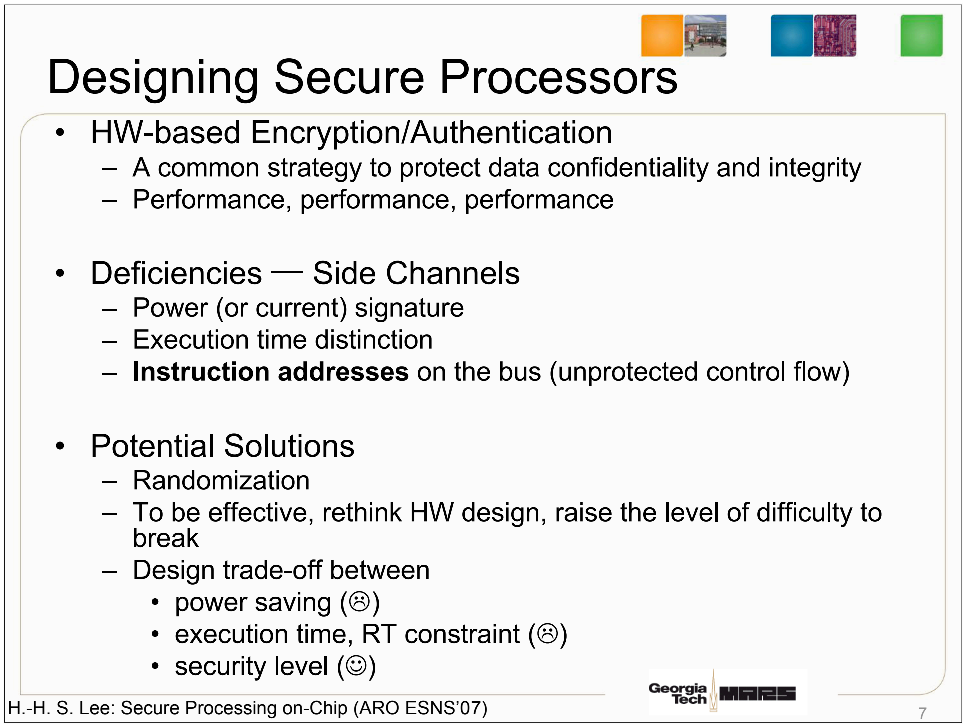
Types of HW-based Physical Attacks

- HW-based physical attacks
  - Trace system bus, peripheral bus
  - Power/Timing analysis
  - Build fake devices, device spoof (e.g., MOD-chip)
  - Modify RAM
  - Replay bus signals, fake bus signal injection

- XBOX with MOD-chip installed. MOD-chip is a low cost bus snoop and spoof device widely used to break XBOX security.

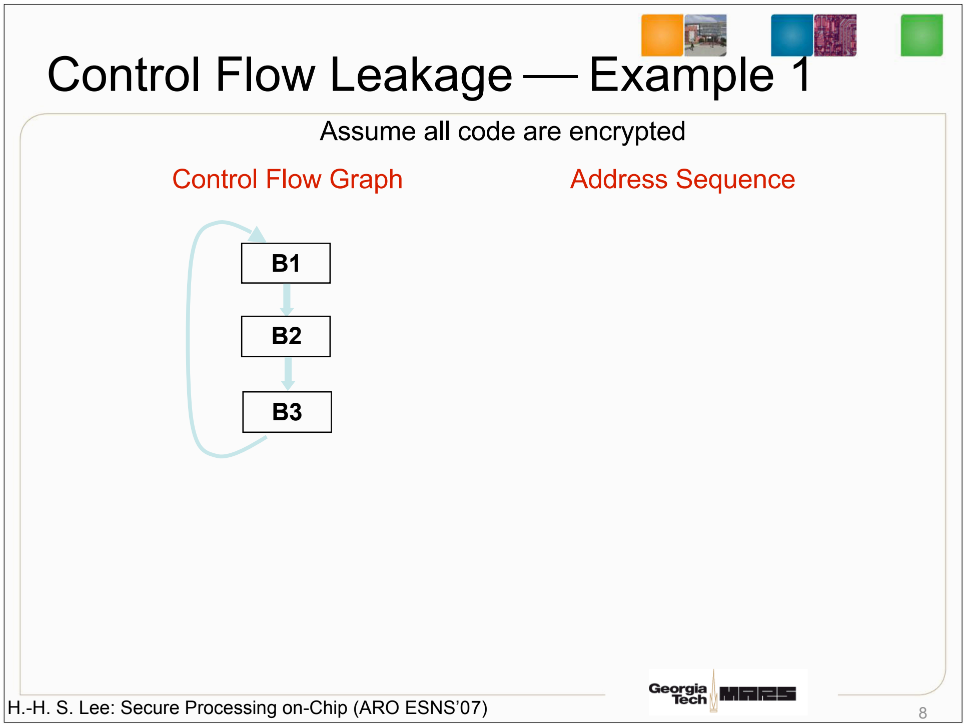
H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

6



Designing Secure Processors

- HW-based Encryption/Authentication
  - A common strategy to protect data confidentiality and integrity
  - Performance, performance, performance
- Deficiencies — Side Channels
  - Power (or current) signature
  - Execution time distinction
  - **Instruction addresses** on the bus (unprotected control flow)
- Potential Solutions
  - Randomization
  - To be effective, rethink HW design, raise the level of difficulty to break
  - Design trade-off between
    - power saving (☹)
    - execution time, RT constraint (☹)
    - security level (☺)

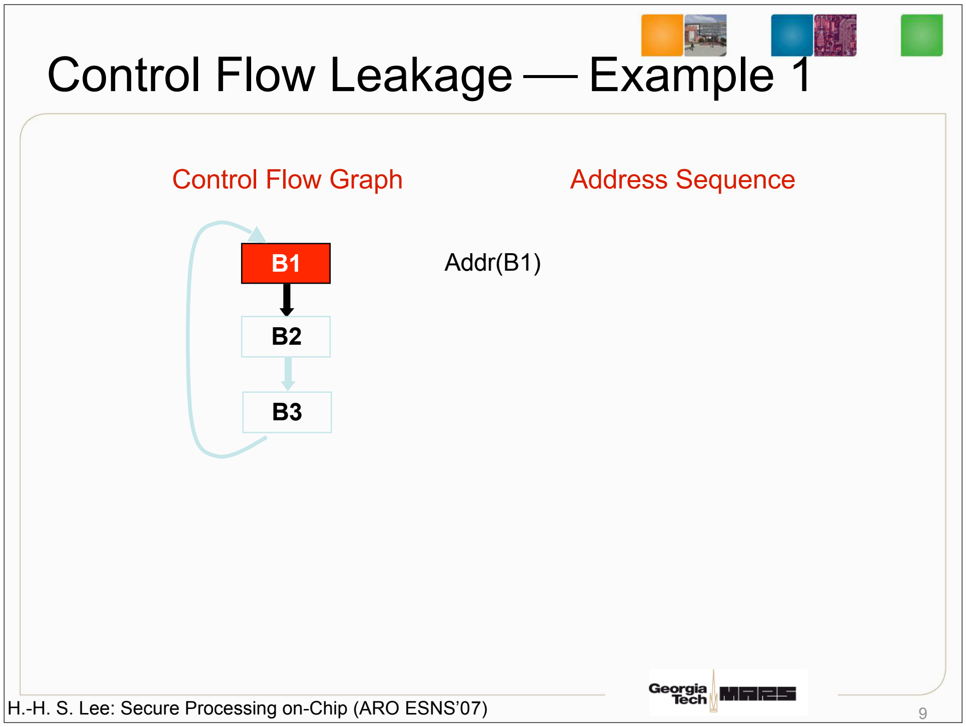


Control Flow Leakage — Example 1

Assume all code are encrypted

Control Flow Graph

Address Sequence



Control Flow Leakage — Example 1

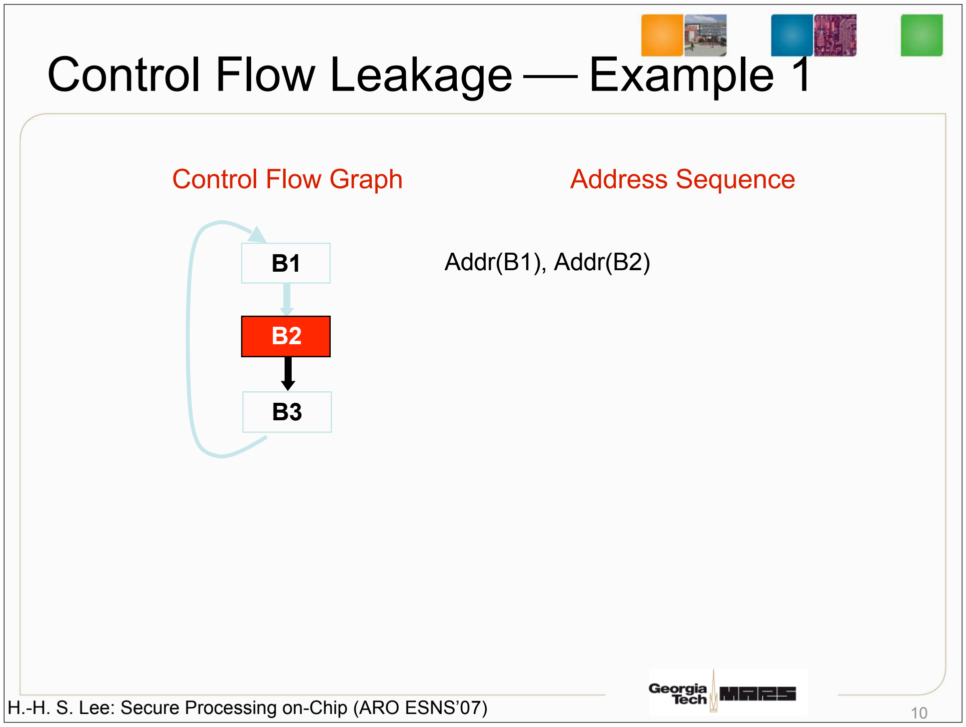
Control Flow Graph

Address Sequence

Addr(B1)

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

9



Control Flow Leakage — Example 1

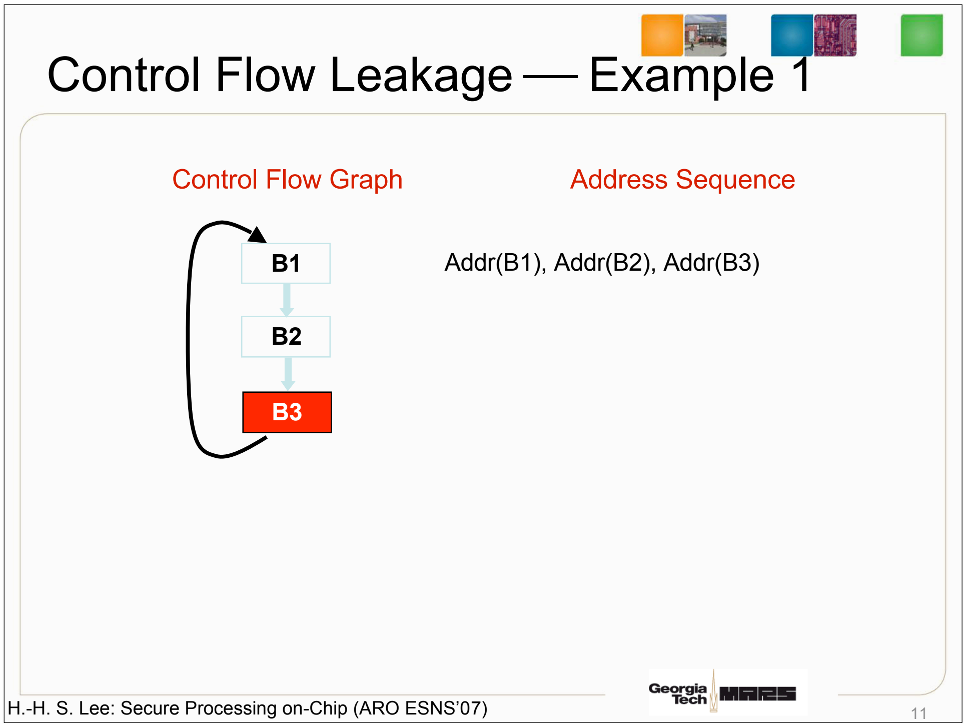
Control Flow Graph

Address Sequence

Addr(B1), Addr(B2)

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

10



Control Flow Leakage — Example 1

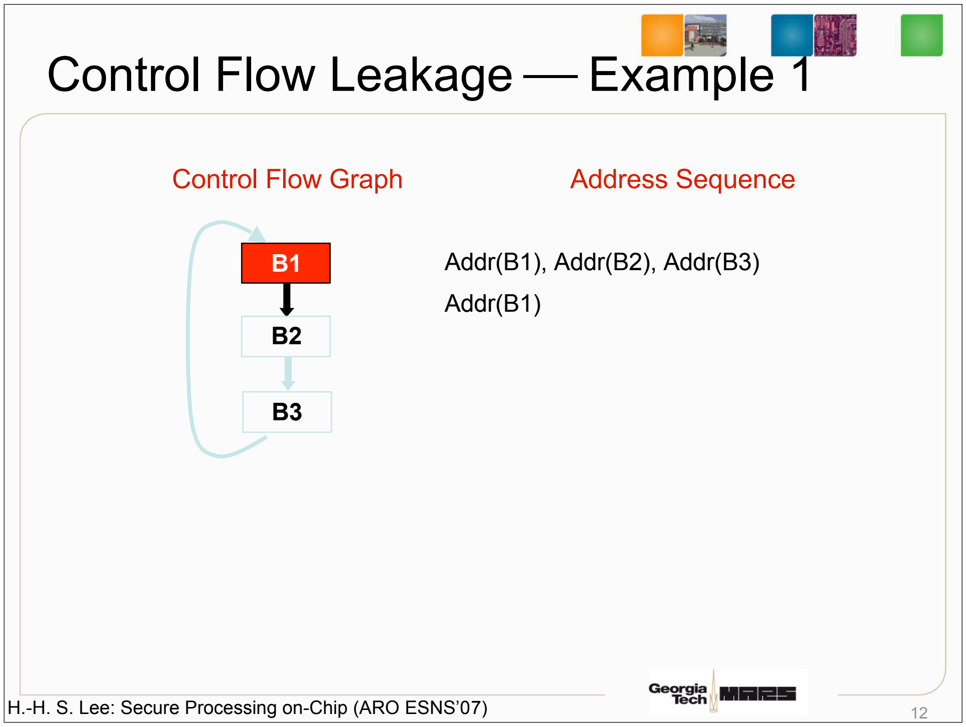
Control Flow Graph

Address Sequence

Addr(B1), Addr(B2), Addr(B3)

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

11



Control Flow Leakage — Example 1

Control Flow Graph

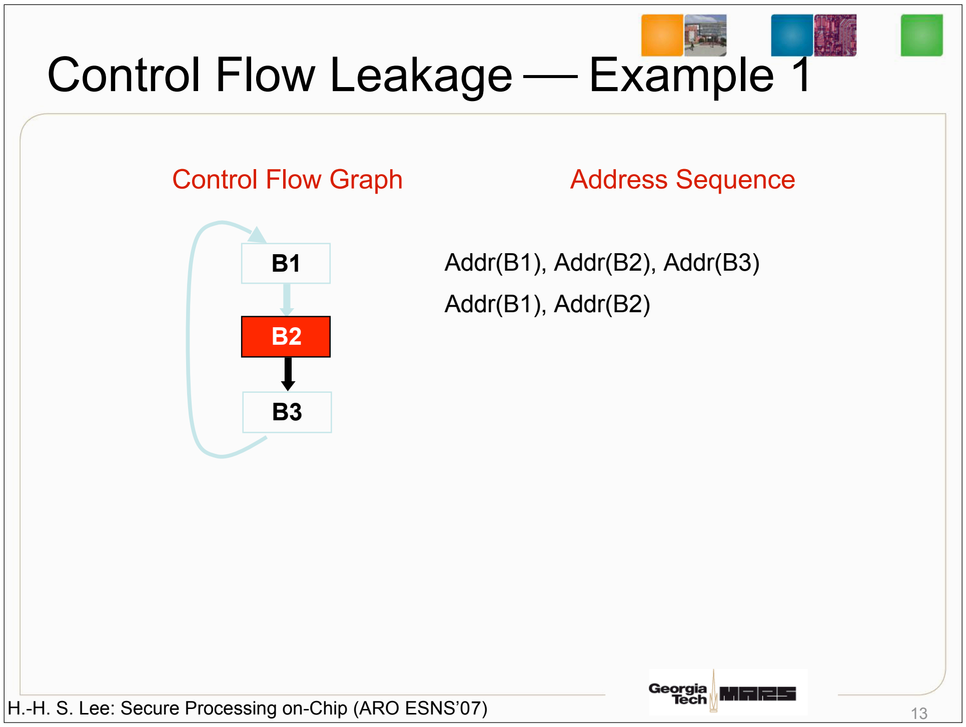
Address Sequence

Addr(B1), Addr(B2), Addr(B3)

Addr(B1)

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

12



Control Flow Leakage — Example 1

Control Flow Graph

B1

B2

B3

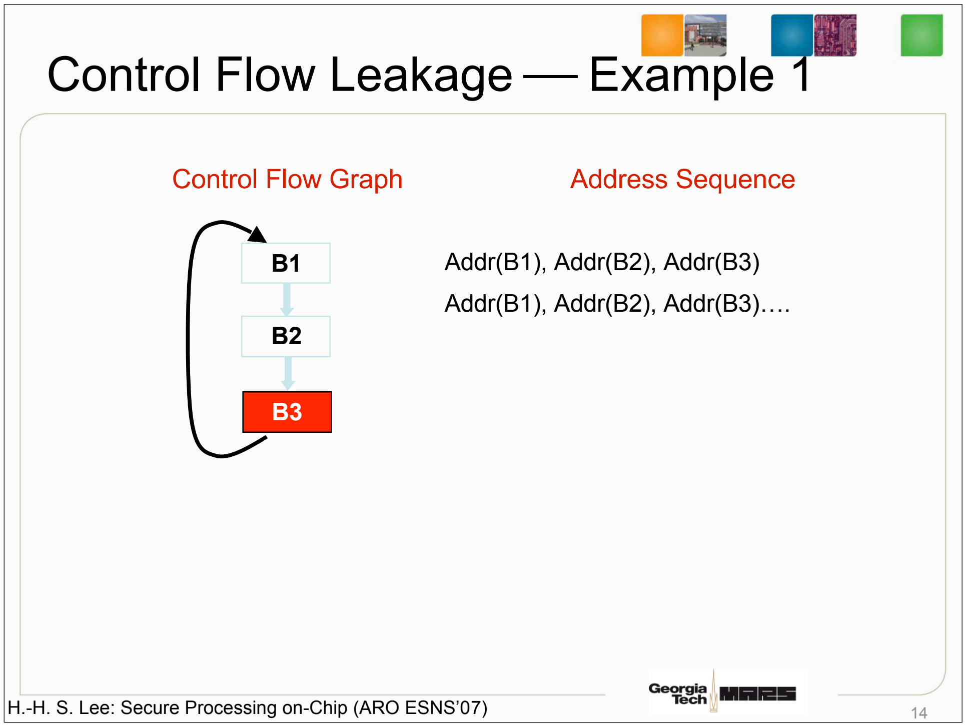
Address Sequence

Addr(B1), Addr(B2), Addr(B3)

Addr(B1), Addr(B2)

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

13



Control Flow Leakage — Example 1

Control Flow Graph

B1

B2

B3

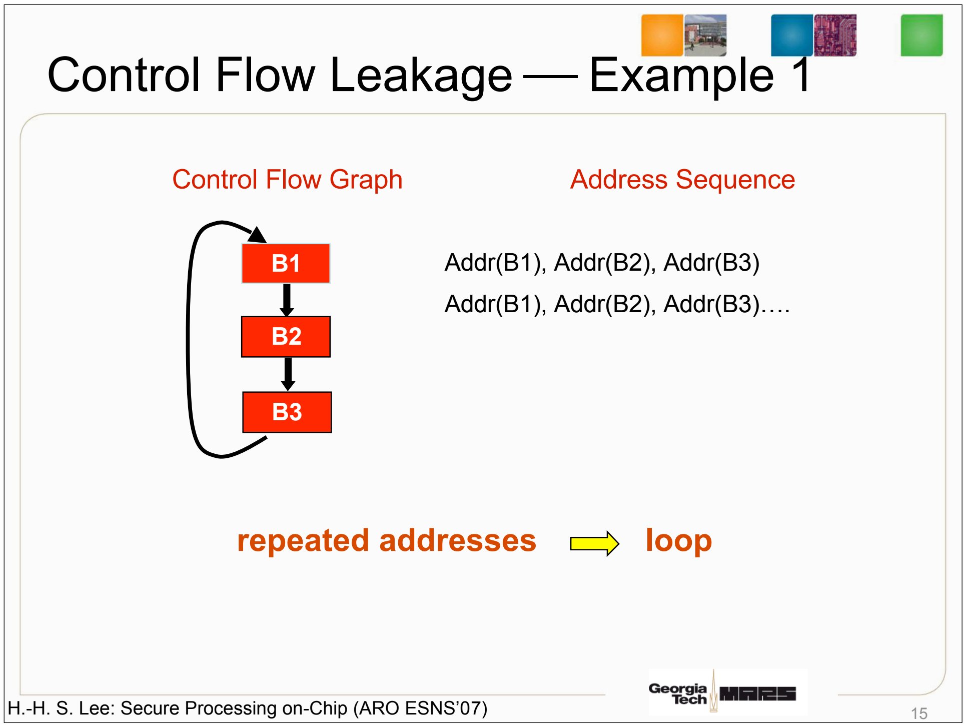
Address Sequence

Addr(B1), Addr(B2), Addr(B3)

Addr(B1), Addr(B2), Addr(B3)….

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

14



Control Flow Leakage — Example 1

Control Flow Graph

Address Sequence

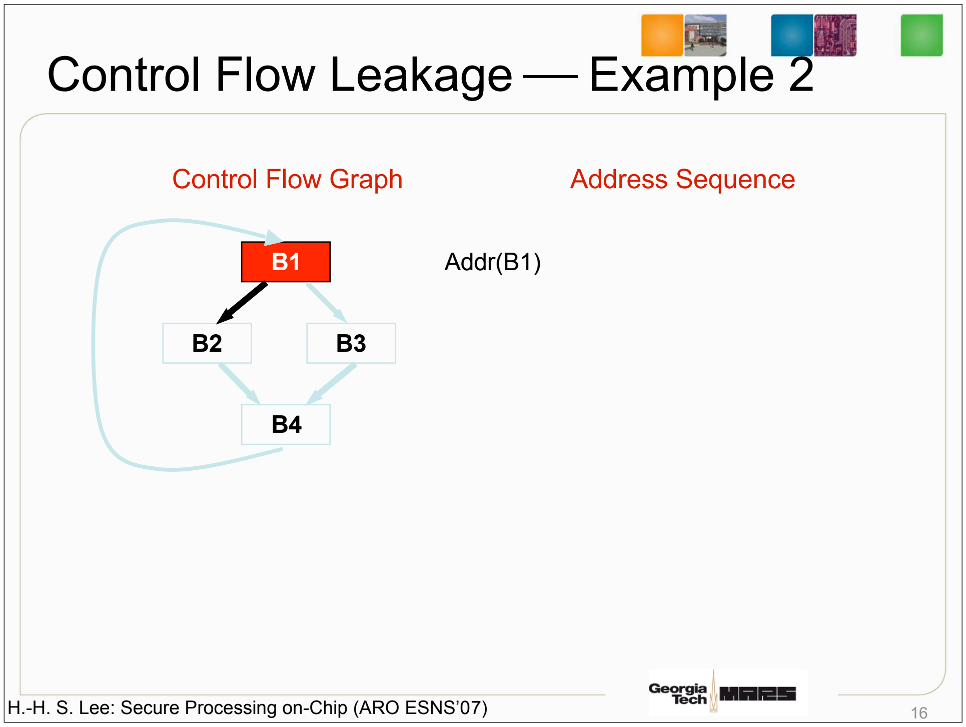
Addr(B1), Addr(B2), Addr(B3)

Addr(B1), Addr(B2), Addr(B3)….

repeated addresses ⇨ loop

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

15



Control Flow Leakage — Example 2

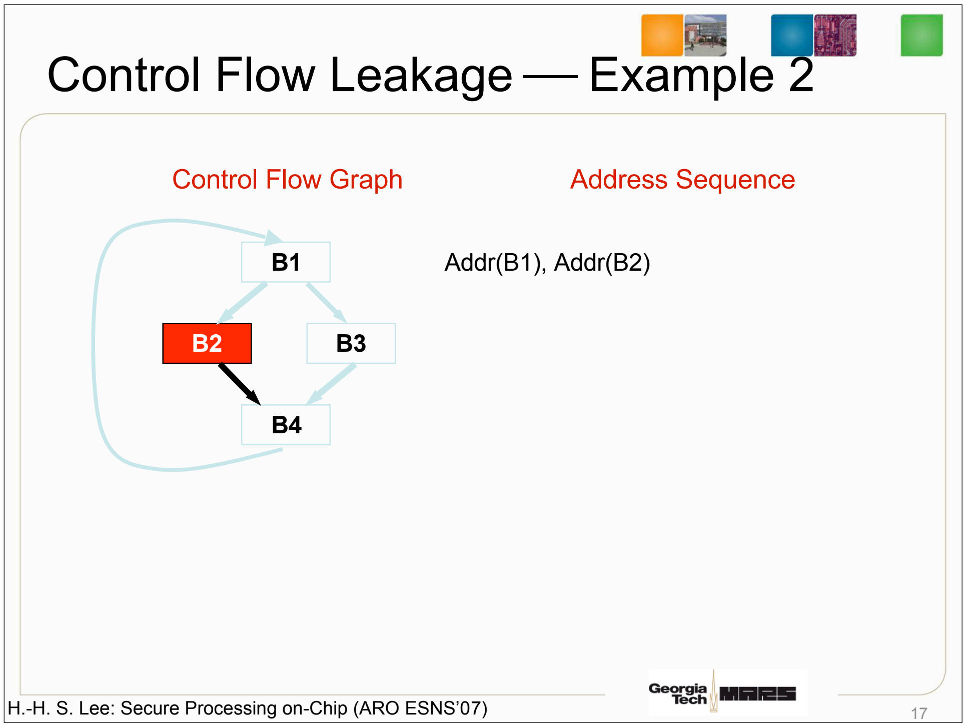
Control Flow Graph

Address Sequence

Addr(B1)

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

16



Control Flow Leakage — Example 2

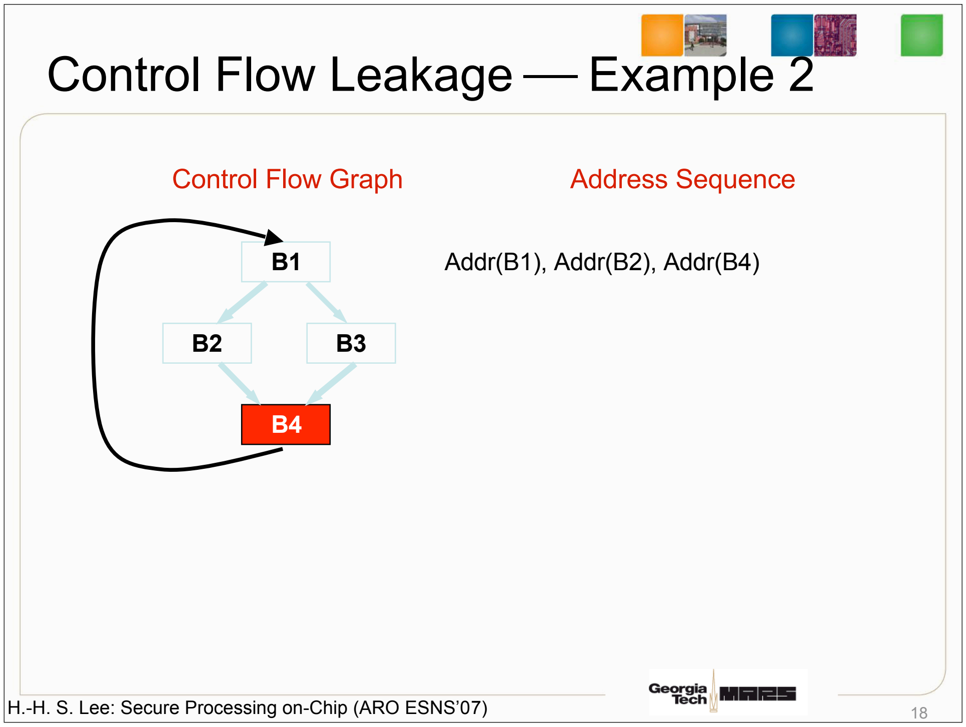
Control Flow Graph

Address Sequence

Addr(B1), Addr(B2)

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

17



Control Flow Leakage — Example 2

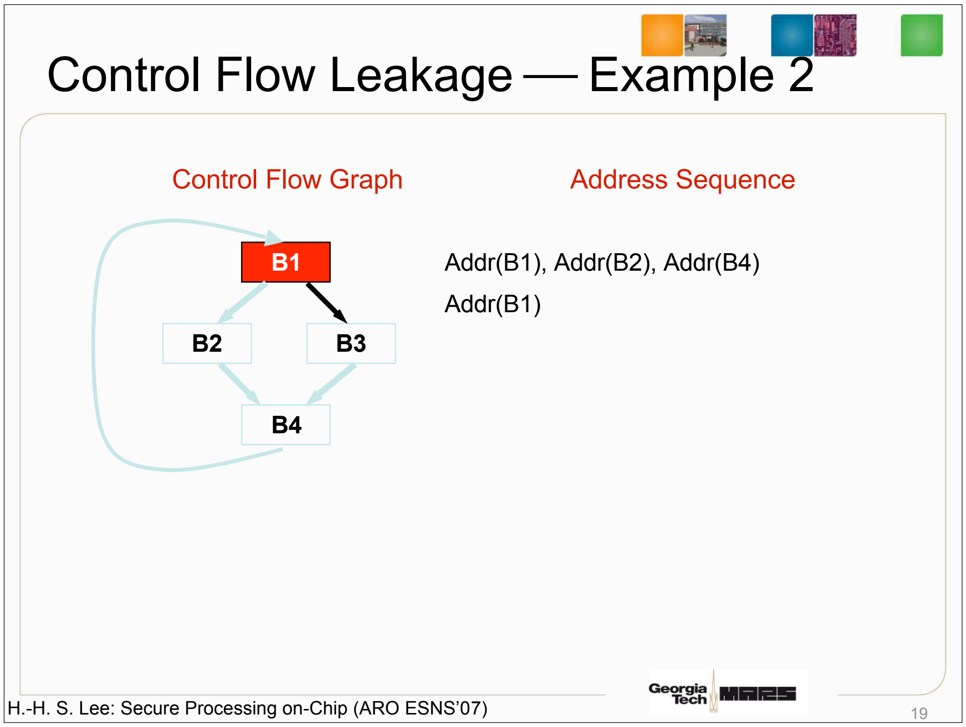
Control Flow Graph

Address Sequence

Addr(B1), Addr(B2), Addr(B4)

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

18



Control Flow Leakage — Example 2

Control Flow Graph

B1

B2

B3

B4

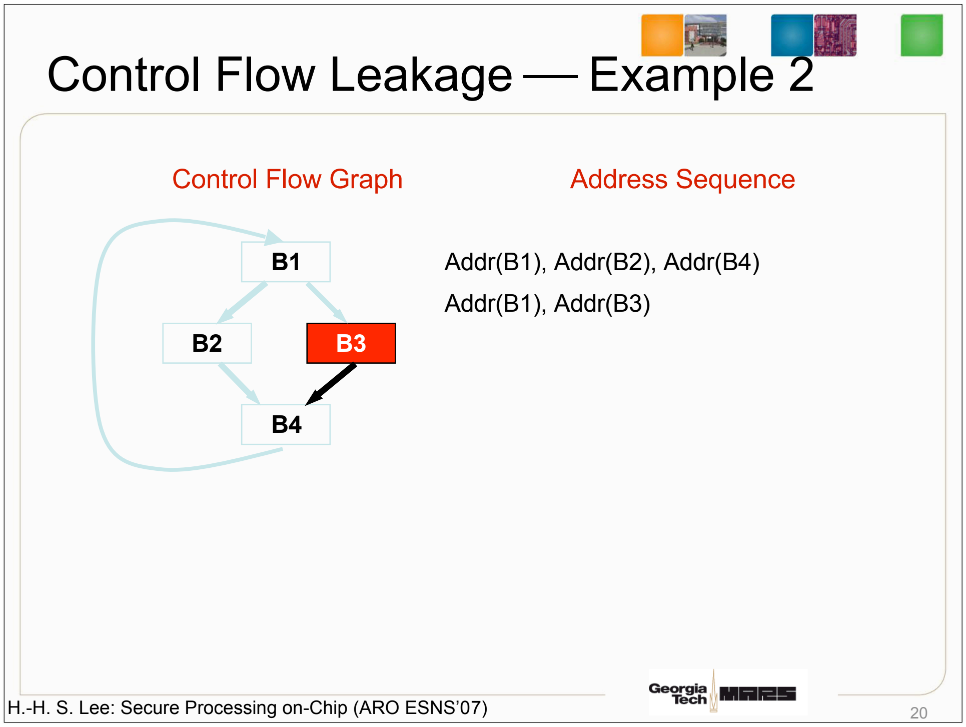
Address Sequence

Addr(B1), Addr(B2), Addr(B4)

Addr(B1)

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

19



Control Flow Leakage — Example 2

Control Flow Graph

B1

B2

B3

B4

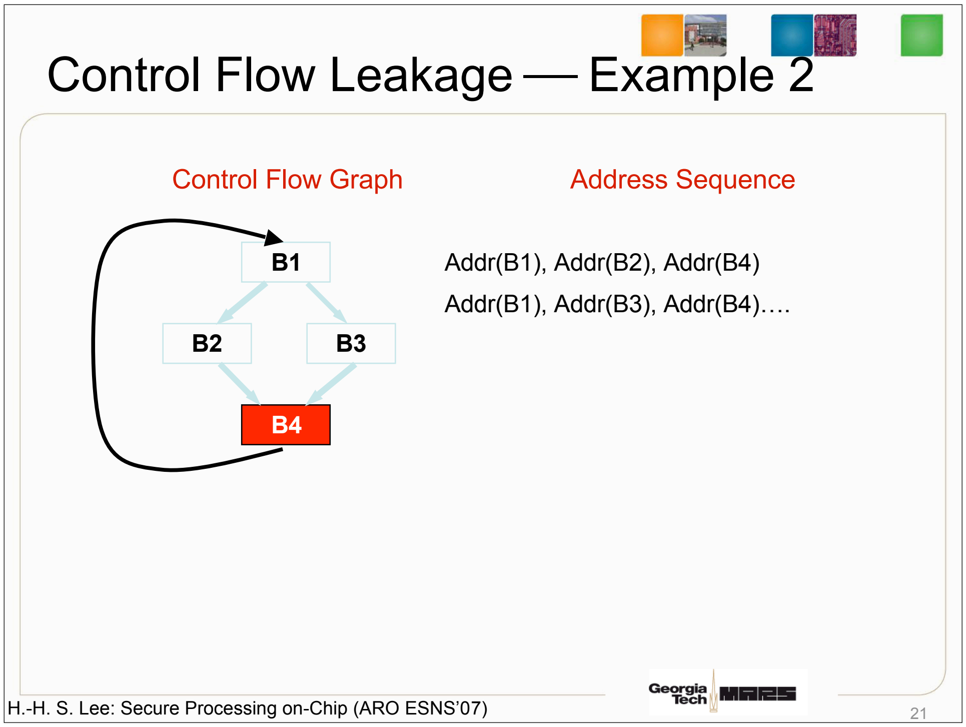
Address Sequence

Addr(B1), Addr(B2), Addr(B4)

Addr(B1), Addr(B3)

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

20



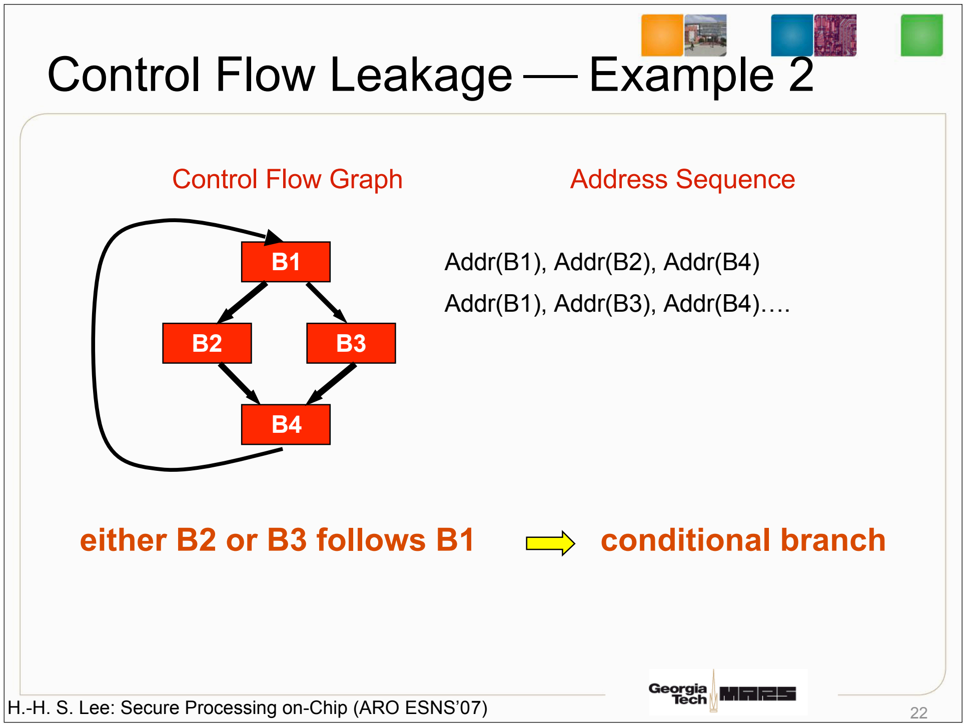
Control Flow Leakage — Example 2

Control Flow Graph

Address Sequence

Addr(B1), Addr(B2), Addr(B4)

Addr(B1), Addr(B3), Addr(B4)….



Control Flow Leakage — Example 2

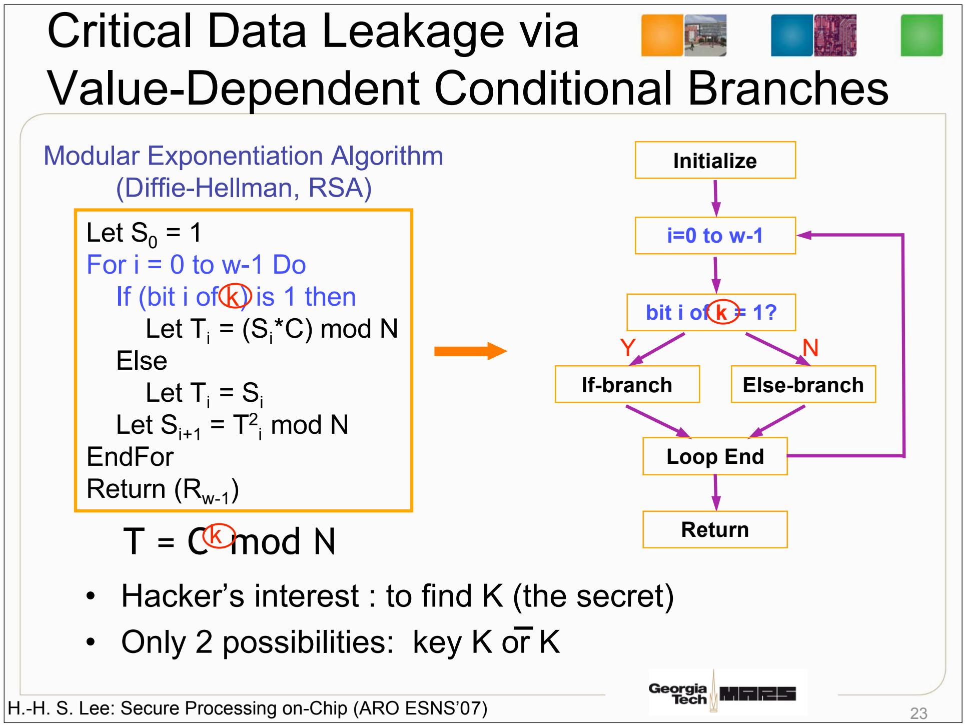
Control Flow Graph

Address Sequence

Addr(B1), Addr(B2), Addr(B4)

Addr(B1), Addr(B3), Addr(B4)….

either B2 or B3 follows B1 ⇨ **conditional branch**



Critical Data Leakage via Value-Dependent Conditional Branches

Modular Exponentiation Algorithm
(Diffie-Hellman, RSA)

```
Let $S_0$ = 1
For i = 0 to w-1 Do
   If (bit i of k) is 1 then
      Let $T_i$ = ($S_i$*C) mod N
   Else
      Let $T_i$ = $S_i$
   Let $S_{i+1}$ = $T^2_i$ mod N
EndFor
Return ($R_{w-1}$)
```

$T = C^k$ mod N

Initialize

i=0 to w-1

bit i of k = 1?

Y

N

If-branch

Else-branch

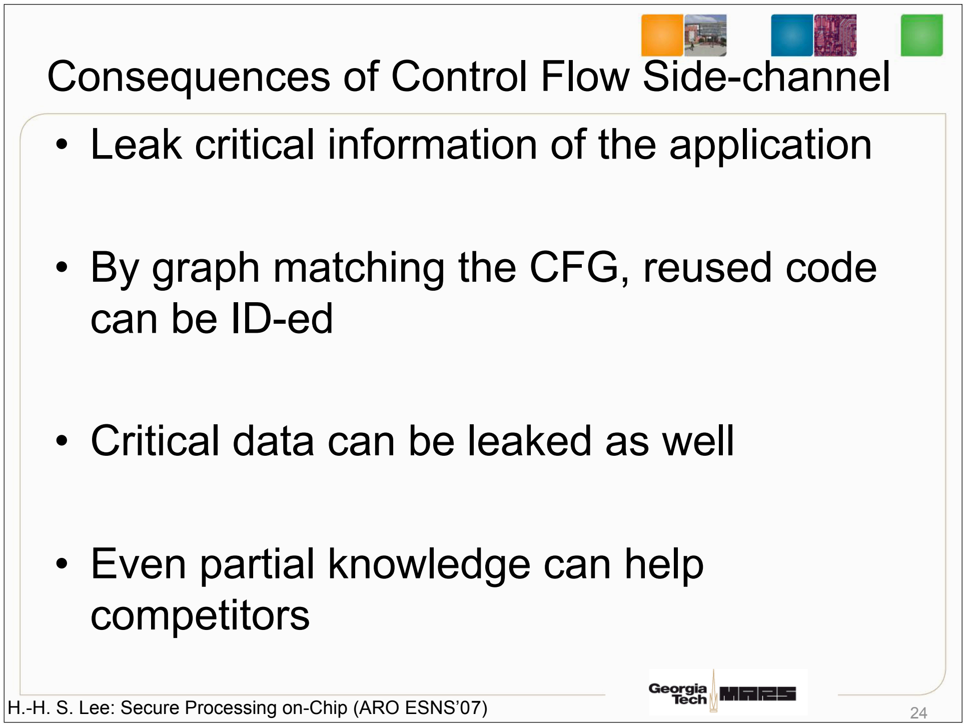
Loop End

Return

- Hacker’s interest : to find K (the secret)
- Only 2 possibilities: key K or $\overline{K}$

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS’07)

23

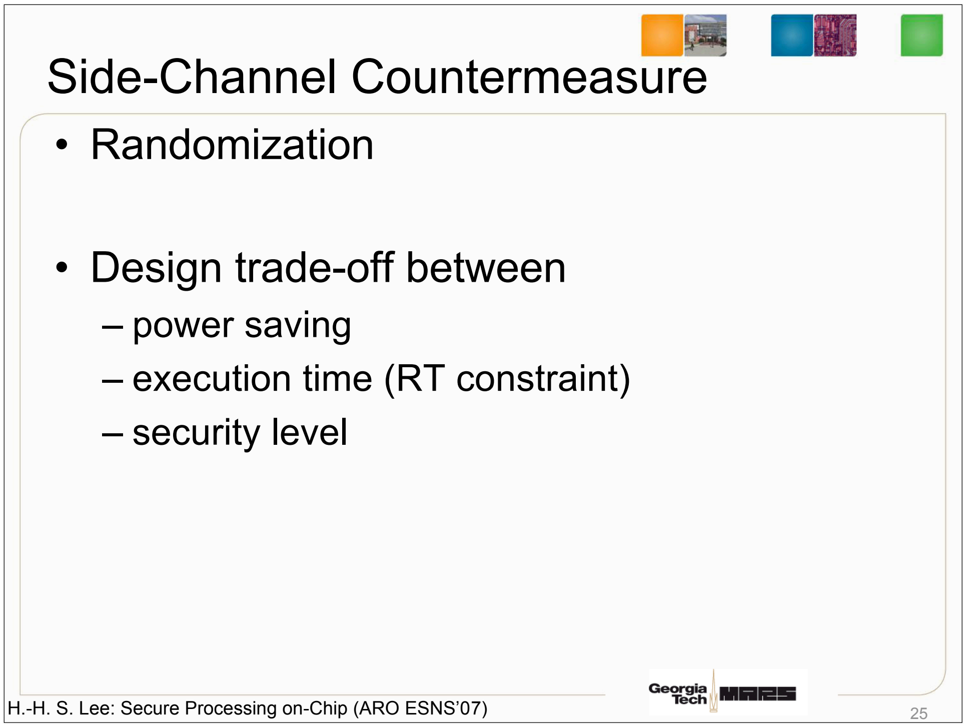


Consequences of Control Flow Side-channel

- Leak critical information of the application
- By graph matching the CFG, reused code can be ID-ed
- Critical data can be leaked as well
- Even partial knowledge can help competitors

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS’07)

24

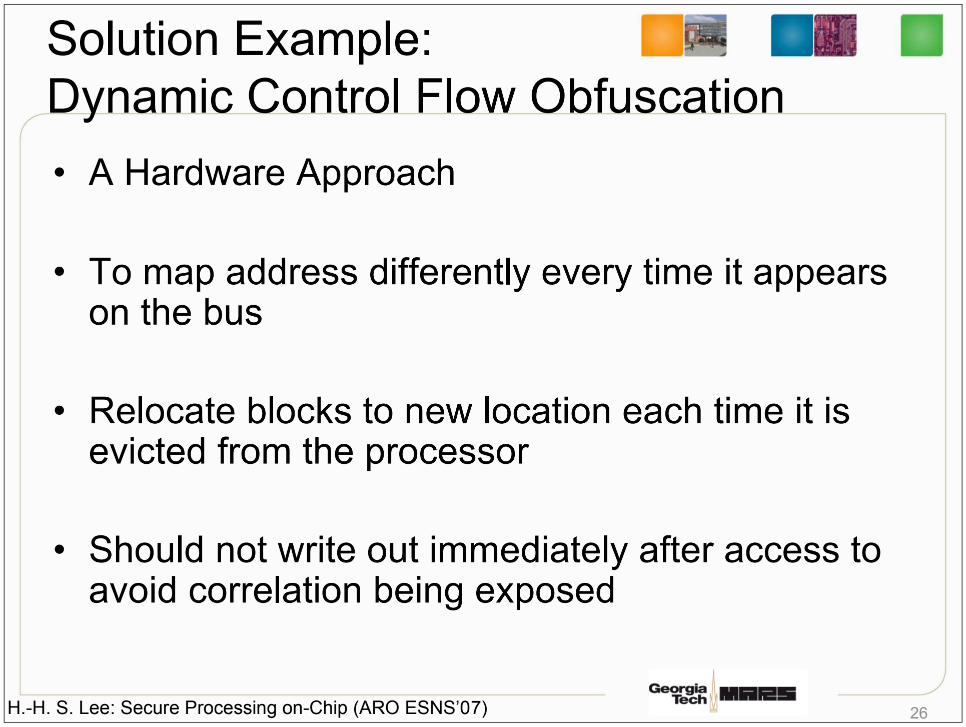


Side-Channel Countermeasure

- Randomization
- Design trade-off between
  - power saving
  - execution time (RT constraint)
  - security level

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

25

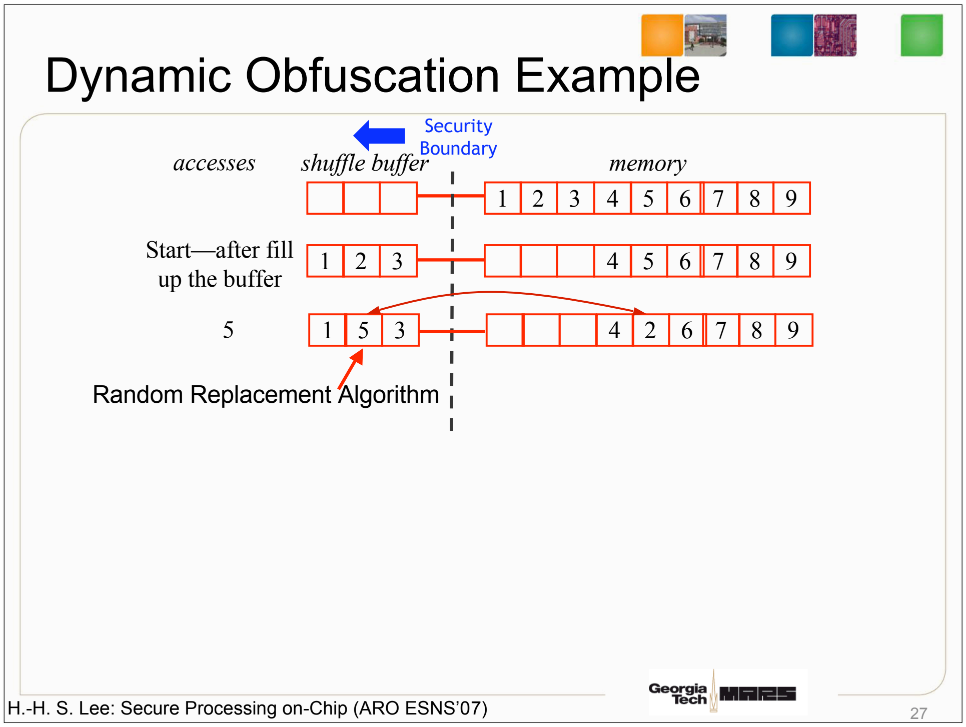


Solution Example: Dynamic Control Flow Obfuscation

- A Hardware Approach
- To map address differently every time it appears on the bus
- Relocate blocks to new location each time it is evicted from the processor
- Should not write out immediately after access to avoid correlation being exposed

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

26



Dynamic Obfuscation Example

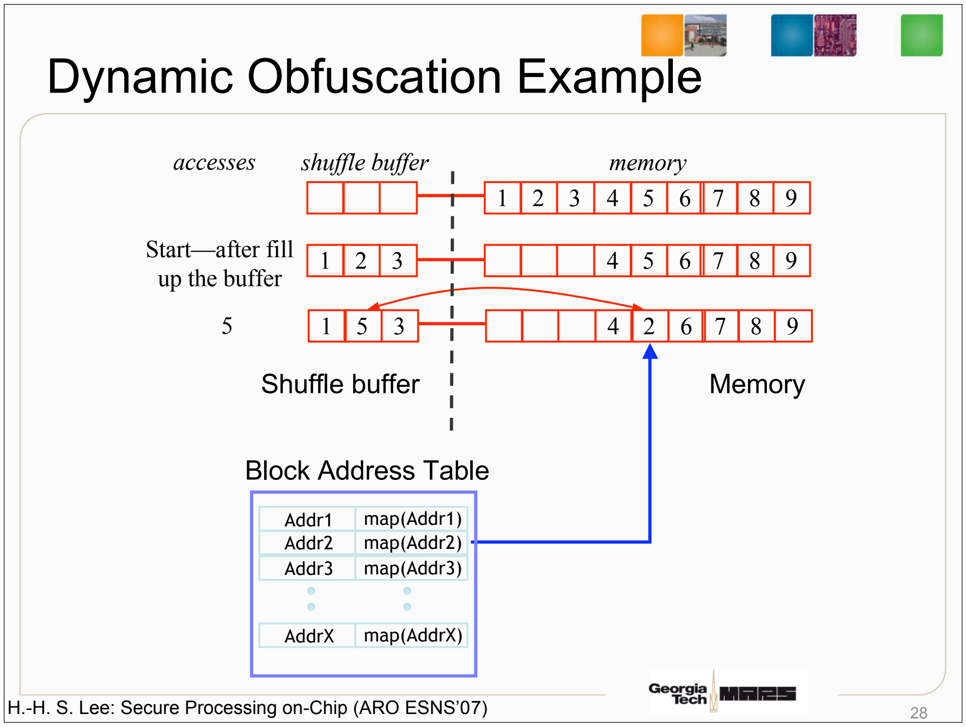
Security Boundary

accesses	shuffle buffer	memory
		1 2 3 4 5 6 7 8 9
Start—after fill up the buffer	1 2 3	4 5 6 7 8 9
5	1 5 3	4 2 6 7 8 9

Random Replacement Algorithm

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

27



Dynamic Obfuscation Example

accesses	shuffle buffer	memory
		1 2 3 4 5 6 7 8 9
Start—after fill up the buffer	1 2 3	4 5 6 7 8 9
5	1 5 3	4 2 6 7 8 9

Shuffle buffer

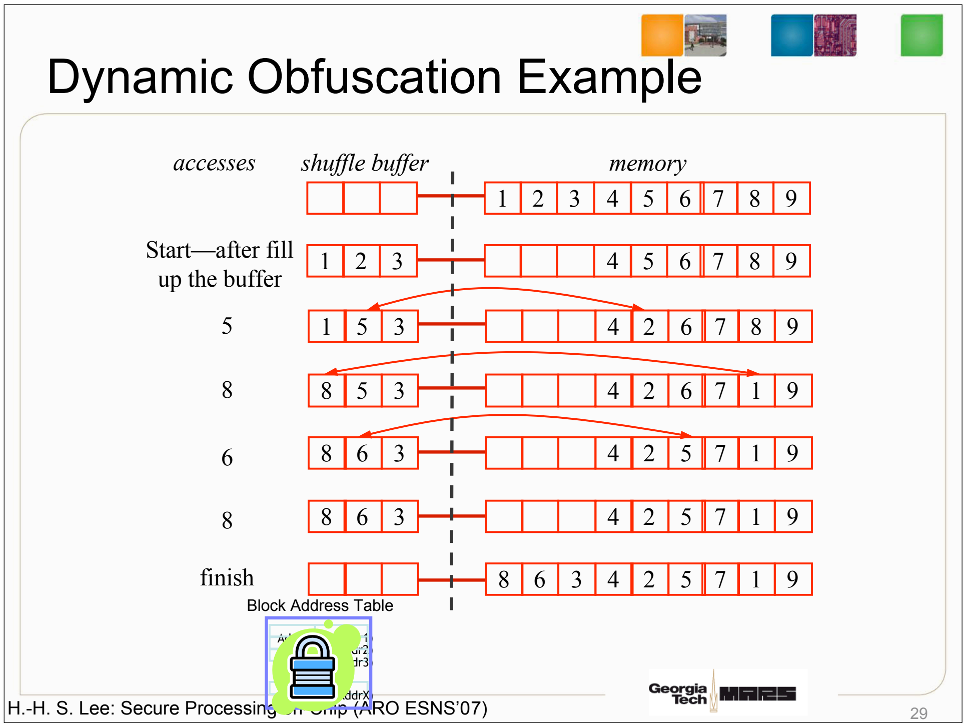
Memory

Block Address Table

Addr1	map(Addr1)
Addr2	map(Addr2)
Addr3	map(Addr3)
•	•
AddrX	map(AddrX)

H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

28



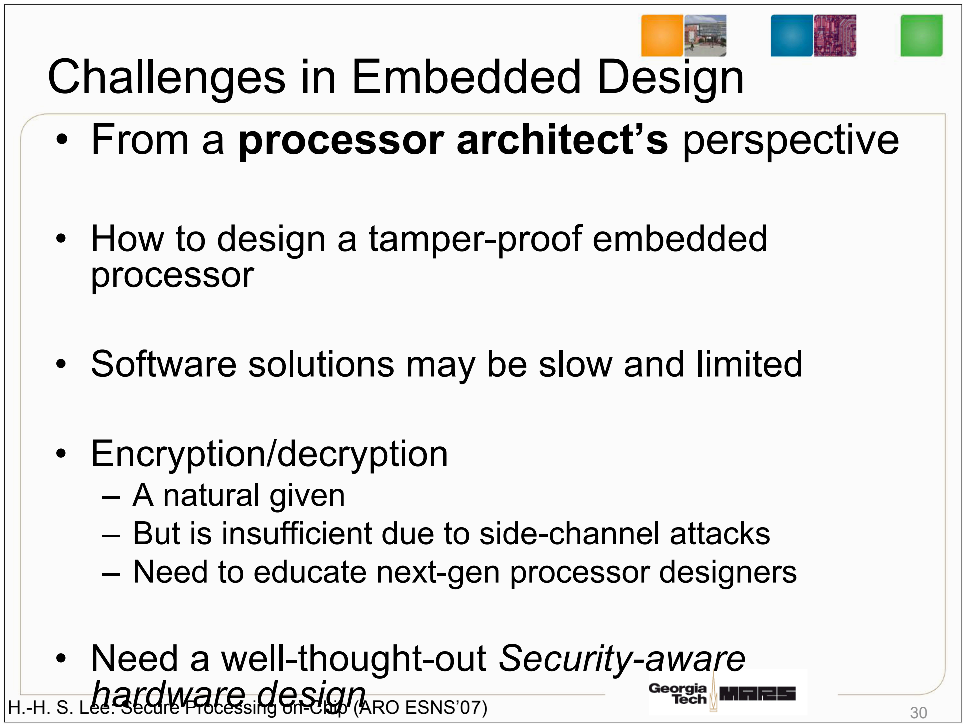
Dynamic Obfuscation Example

accesses	shuffle buffer			memory								
				1	2	3	4	5	6	7	8	9
Start—after fill up the buffer	1	2	3				4	5	6	7	8	9
5	1	5	3				4	2	6	7	8	9
8	8	5	3				4	2	6	7	1	9
6	8	6	3				4	2	5	7	1	9
8	8	6	3				4	2	5	7	1	9
finish				8	6	3	4	2	5	7	1	9

Block Address Table

H.-H. S. Lee: Secure Processing on Chip (ARO ESNS'07)

29

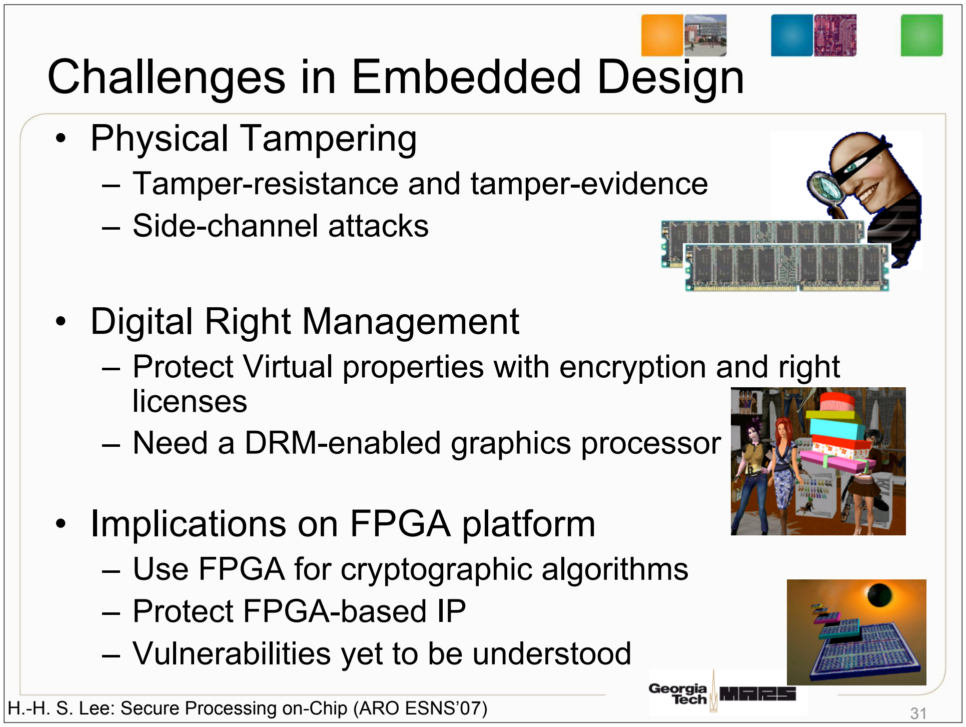


Challenges in Embedded Design

- From a **processor architect's** perspective
- How to design a tamper-proof embedded processor
- Software solutions may be slow and limited
- Encryption/decryption
  - A natural given
  - But is insufficient due to side-channel attacks
  - Need to educate next-gen processor designers
- Need a well-thought-out *Security-aware hardware design*

H.-H. S. Lee: Secure Processing on Chip (ARO ESNS'07)

30



Challenges in Embedded Design

- Physical Tampering
  - Tamper-resistance and tamper-evidence
  - Side-channel attacks
- Digital Right Management
  - Protect Virtual properties with encryption and right licenses
  - Need a DRM-enabled graphics processor
- Implications on FPGA platform
  - Use FPGA for cryptographic algorithms
  - Protect FPGA-based IP
  - Vulnerabilities yet to be understood

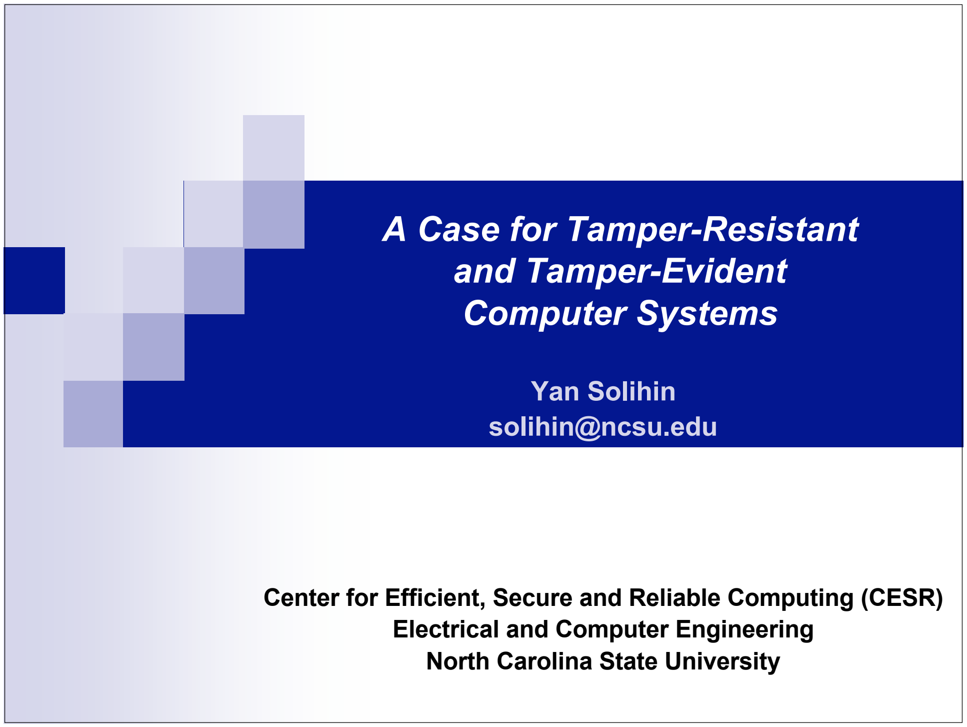
H.-H. S. Lee: Secure Processing on-Chip (ARO ESNS'07)

31

Thank You!

Home of the 1996 Olympic Village
Georgia Tech

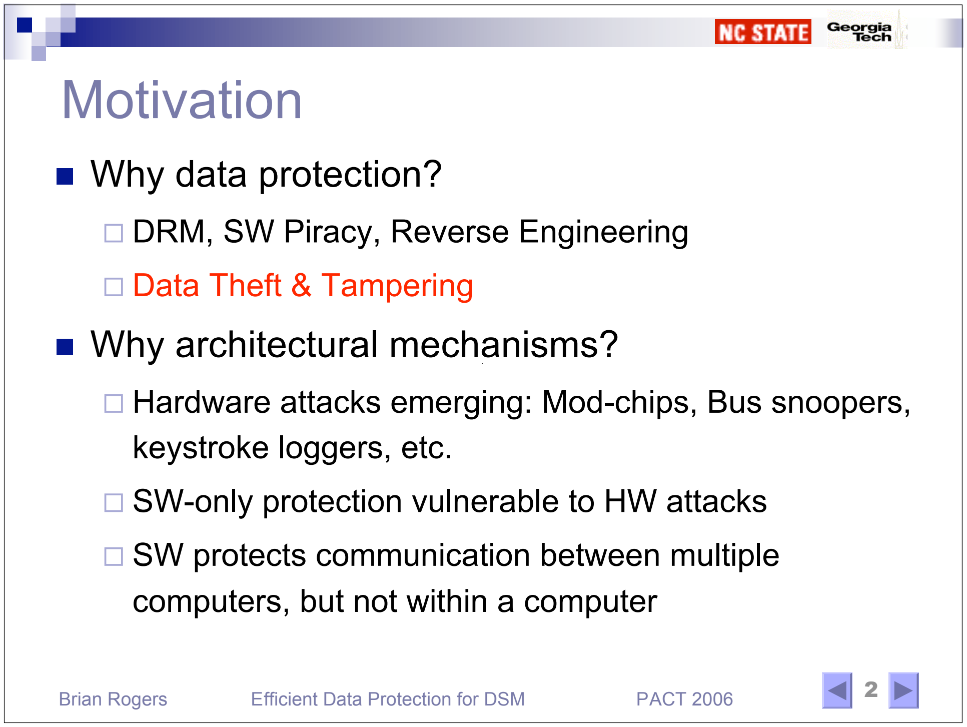
http://arch.ece.gatech.edu



A Case for Tamper-Resistant and Tamper-Evident Computer Systems

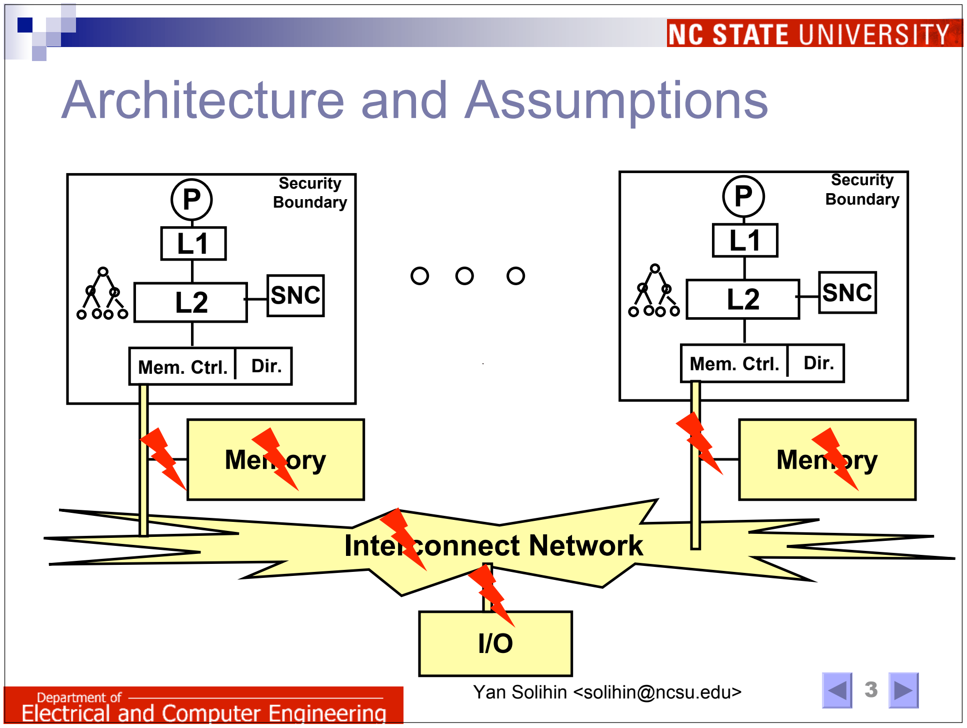
Yan Solihin
solihin@ncsu.edu

Center for Efficient, Secure and Reliable Computing (CESR)
Electrical and Computer Engineering
North Carolina State University



Motivation

- Why data protection?
  - □ DRM, SW Piracy, Reverse Engineering
  - □ Data Theft & Tampering
- Why architectural mechanisms?
  - □ Hardware attacks emerging: Mod-chips, Bus snoopers, keystroke loggers, etc.
  - □ SW-only protection vulnerable to HW attacks
  - □ SW protects communication between multiple computers, but not within a computer

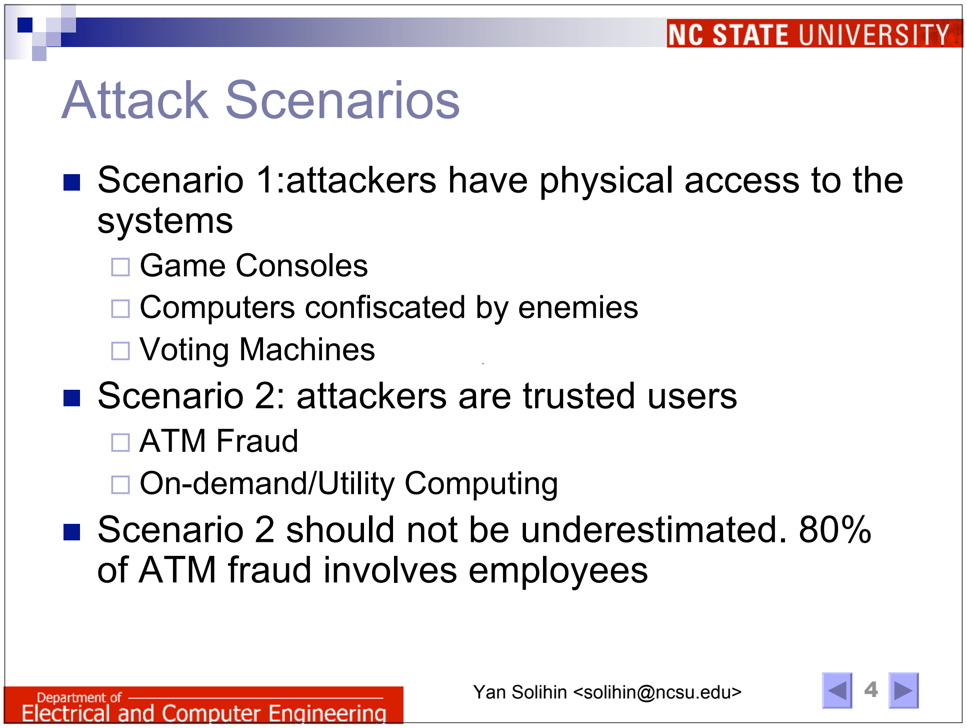


Architecture and Assumptions

P
L1
L2
SNC
Mem. Ctrl.
Dir.
Security Boundary

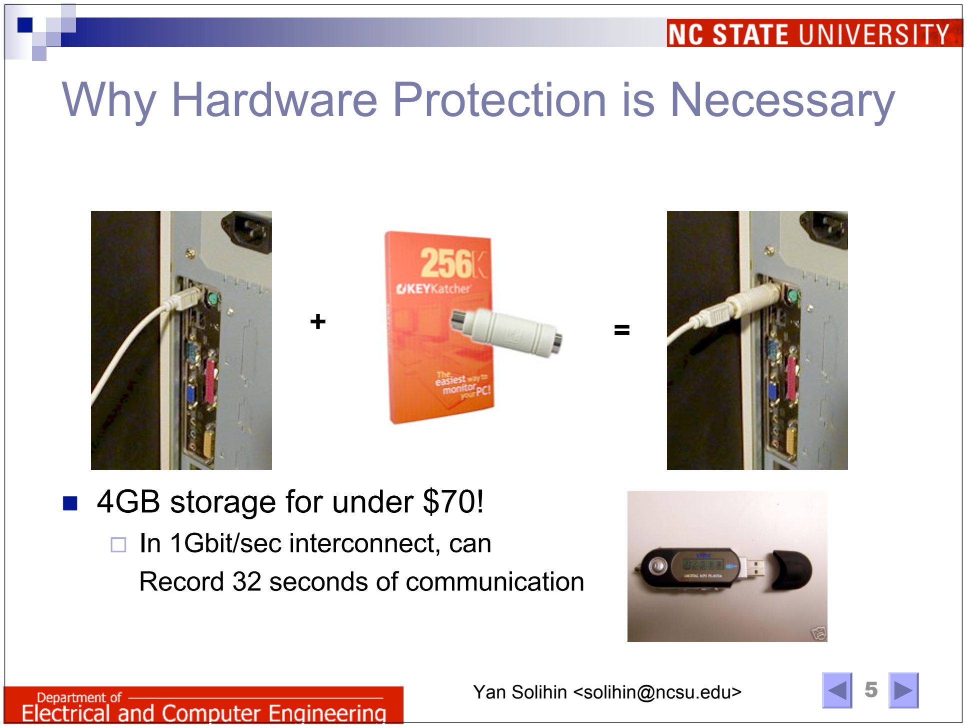
P
L1
L2
SNC
Mem. Ctrl.
Dir.
Security Boundary

Memory
Memory
Interconnect Network
I/O



Attack Scenarios

- Scenario 1:attackers have physical access to the systems
  - Game Consoles
  - Computers confiscated by enemies
  - Voting Machines
- Scenario 2: attackers are trusted users
  - ATM Fraud
  - On-demand/Utility Computing
- Scenario 2 should not be underestimated. 80% of ATM fraud involves employees

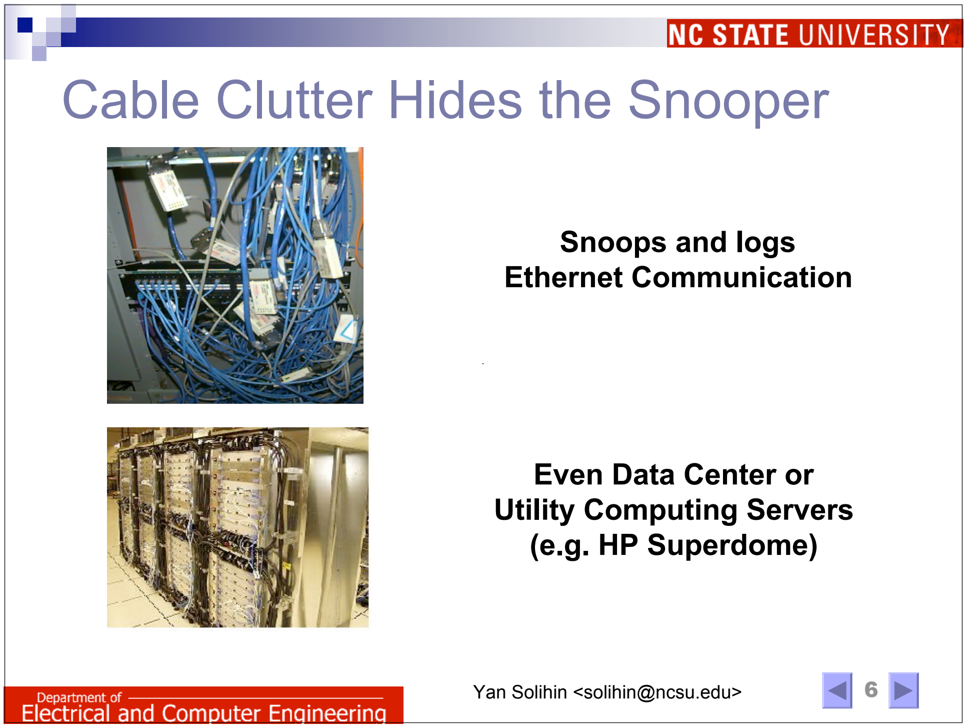


Why Hardware Protection is Necessary

+ =

256K
KEYKatcher
The easiest way to monitor your PC!

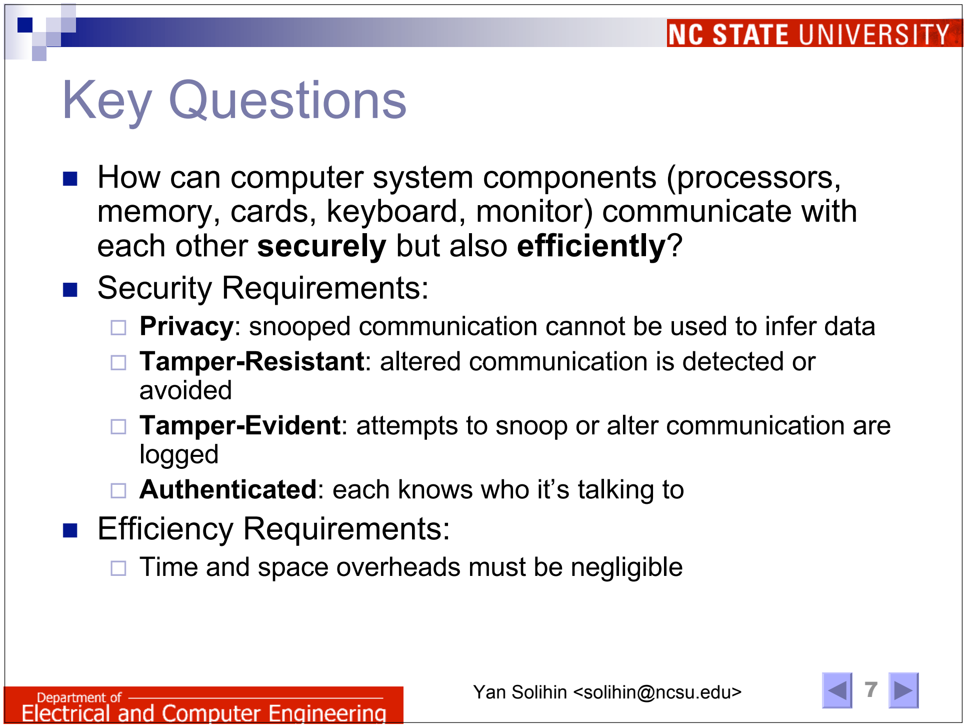
- 4GB storage for under $70!
  - □ In 1Gbit/sec interconnect, can Record 32 seconds of communication



Cable Clutter Hides the Snooper

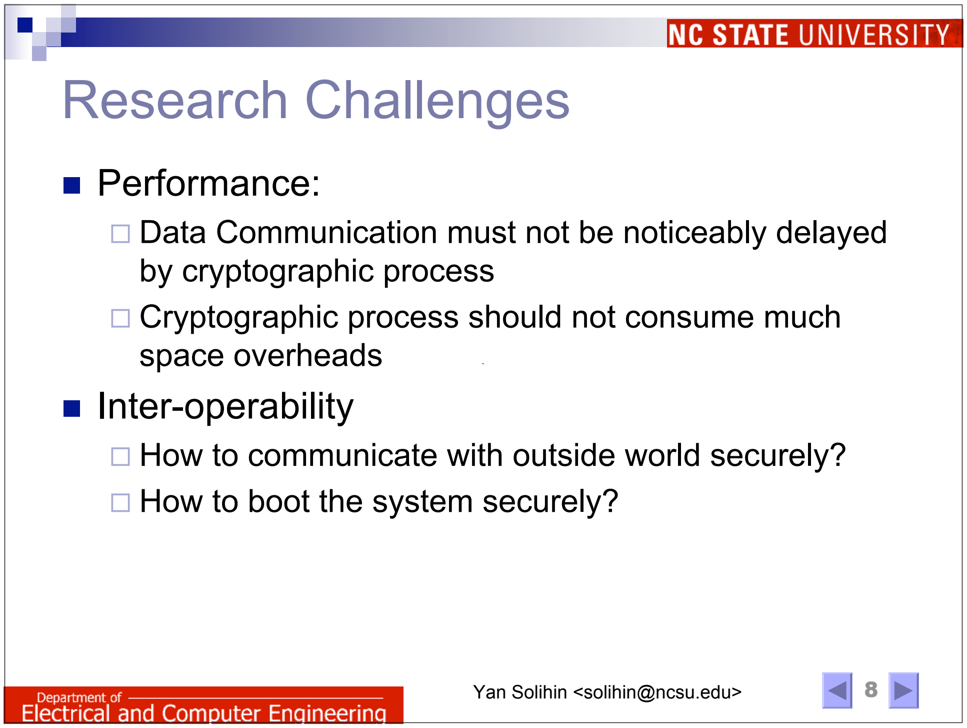
Snoops and logs Ethernet Communication

Even Data Center or Utility Computing Servers (e.g. HP Superdome)



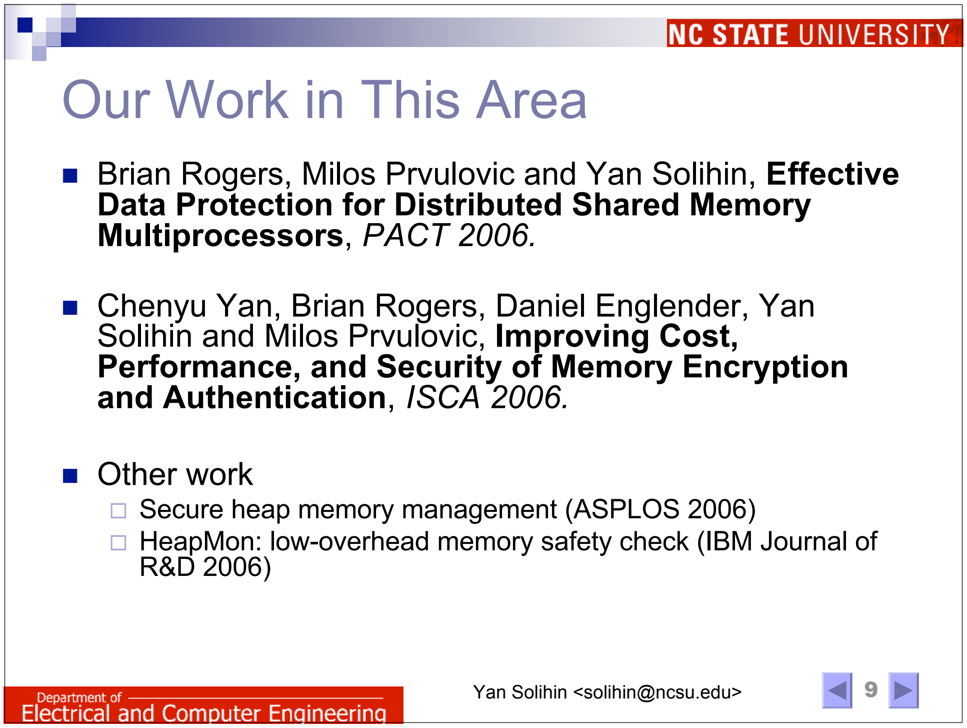
Key Questions

- How can computer system components (processors, memory, cards, keyboard, monitor) communicate with each other **securely** but also **efficiently**?
- Security Requirements:
  - □ **Privacy**: snooped communication cannot be used to infer data
  - □ **Tamper-Resistant**: altered communication is detected or avoided
  - □ **Tamper-Evident**: attempts to snoop or alter communication are logged
  - □ **Authenticated**: each knows who it's talking to
- Efficiency Requirements:
  - □ Time and space overheads must be negligible



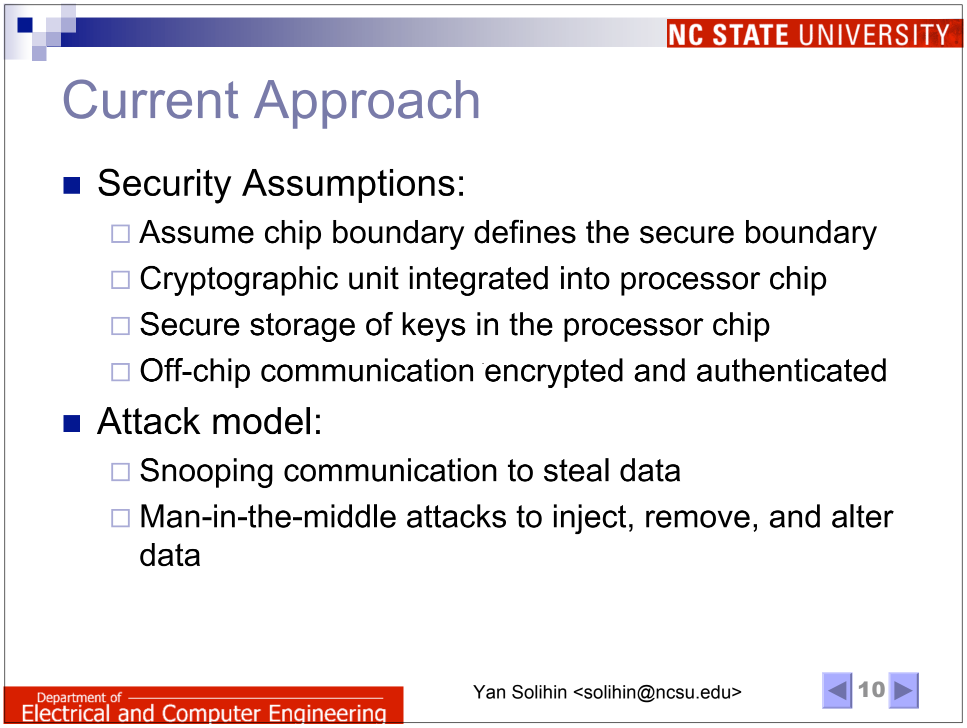
Research Challenges

- Performance:
  - □ Data Communication must not be noticeably delayed by cryptographic process
  - □ Cryptographic process should not consume much space overheads
- Inter-operability
  - □ How to communicate with outside world securely?
  - □ How to boot the system securely?



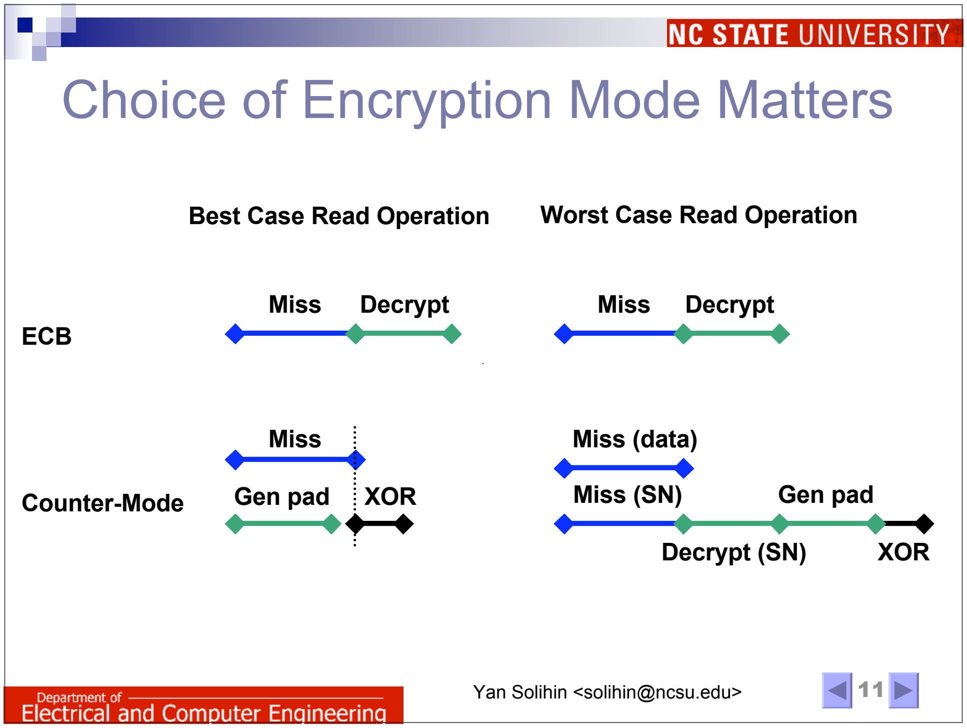
Our Work in This Area

- Brian Rogers, Milos Prvulovic and Yan Solihin, **Effective Data Protection for Distributed Shared Memory Multiprocessors**, *PACT 2006.*
- Chenyu Yan, Brian Rogers, Daniel Englender, Yan Solihin and Milos Prvulovic, **Improving Cost, Performance, and Security of Memory Encryption and Authentication**, *ISCA 2006.*
- Other work
  - □ Secure heap memory management (ASPLOS 2006)
  - □ HeapMon: low-overhead memory safety check (IBM Journal of R&D 2006)



Current Approach

- Security Assumptions:
  - □ Assume chip boundary defines the secure boundary
  - □ Cryptographic unit integrated into processor chip
  - □ Secure storage of keys in the processor chip
  - □ Off-chip communication encrypted and authenticated
- Attack model:
  - □ Snooping communication to steal data
  - □ Man-in-the-middle attacks to inject, remove, and alter data



Choice of Encryption Mode Matters

Best Case Read Operation

Worst Case Read Operation

ECB

Miss

Decrypt

Miss

Decrypt

Counter-Mode

Miss

Gen pad

XOR

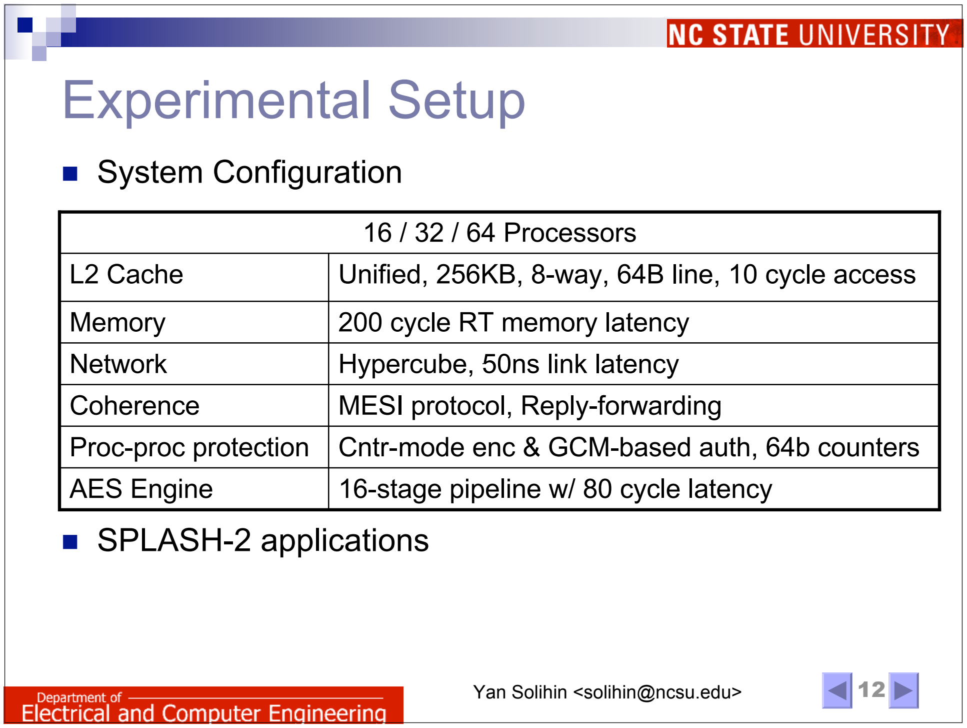
Miss (data)

Miss (SN)

Gen pad

Decrypt (SN)

XOR



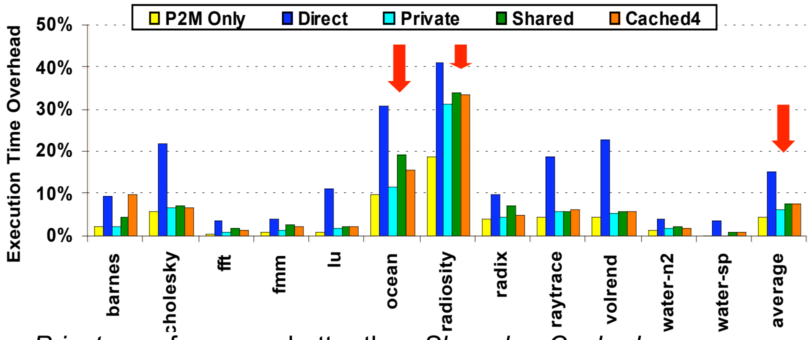
Experimental Setup

- System Configuration

16 / 32 / 64 Processors	
L2 Cache	Unified, 256KB, 8-way, 64B line, 10 cycle access
Memory	200 cycle RT memory latency
Network	Hypercube, 50ns link latency
Coherence	MESI protocol, Reply-forwarding
Proc-proc protection	Cntr-mode enc & GCM-based auth, 64b counters
AES Engine	16-stage pipeline w/ 80 cycle latency

- SPLASH-2 applications

DSM Data Protection Overhead



- *Private:* performance better than *Shared* or *Cached*
- *Cached:* good tradeoff between performance, storage, and scalability
- DSM protection adds only 1% to 3% overhead on average compared to CPU-Mem protection; total overhead is only 6% to 8%

Conclusions

- Hardware Attacks feasible in certain scenarios
- Secure off-chip communication possible with low overheads
- There are many remaining challenges

Session V – Robust Distributed Services

Session Chair: Cliff Wang

ARO Planning Workshop
Security of Embedded Systems and Networks

John A. Stankovic
University of Virginia
January 25, 2005

Position Statement

Wireless sensor networks (WSNs) are composed of large numbers of minimal capacity sensing, computing, and communicating devices. These devices operate in complex and noisy environments. Transient and permanent random failures are commonplace. The considerable redundancy in such systems creates great potential for designing them to continue to provide their specified services even in the face of large numbers of such failures. WSNs are also susceptible to malicious, non-random *security* attacks. For example, a wireless sensor network deployed in remote regions to detect and classify targets could be rendered inoperative by various security attacks. To meet realistic requirements, WSNs must be able to continue to operate satisfactorily in the presence of, and to recover effectively from, security attacks. We propose that safe self-healing and adaptive infrastructures can work together to permit WSNs to continue to operate and self-heal in the presence of failures and security attacks.

Key Limitations and Challenges

1. Many current security solutions are heavyweight. Lightweight versions of current solutions and entirely new, but lightweight solutions are required for WSNs.
2. Many current solutions are not reactive; they attempt to provide security all-the-time. This is often not practical. We require new architectures and systems that can support dynamic reaction to attacks.
3. Detecting attacks is a serious problem in WSNs because of the limited resources that can be assigned to detection and because of the very noisy and failure prone devices, making it more difficult to distinguish between faults and security attacks. This includes stealthy denial of service attacks. The challenge is to solve these problems with new hardware and software solutions.

Promising Innovations

The confluence of four technologies is creating an opportunity for a major new approach to solving some aspects of the security problems for wireless sensor networks. Note that detecting attacks is not addressed in this position statement due to lack of space.

First, WSNs have now evolved to where there are large numbers of decentralized protocols integrated into functioning systems. However, these protocols have not typically addressed security. Large numbers of small sensor nodes each executing decentralized control protocols can provide a basis for new solutions that allow operation in the presence of attacks. For example, VigilNet, the system we developed under the Darpa NEST contract has well over 20 protocols operating in a decentralized manner and can form the basis for (initially) masking security threats; subsequently, based on concepts described below corrupted data or software components are self-healed. In other words, if designed with security in mind, decentralized protocols can prevent attacks in one part of the network from affecting the entire system. We consider the aggregate behavior portrayed by large numbers of decentralized entities a critical technology, but it now needs to be "applied" to security problems.

Second, self-healing technology has progressed in the mainframe and Internet worlds to a degree where some of these ideas can now migrate to WSNs (subject to new sensor network constraints). Self-healing provides a means for masking and repairing security attacks. We suggest a need for extensions to self-healing technology that ascertain if the self-healing actions are safe (first off-line and then on-line) before they are placed into effect. The safety is ascertained by checking that dynamic integration of components to heal problems caused by security attacks meets required security, integrity and interface requirements. A key challenge is to ensure that the self-healing itself does not introduce new security vulnerabilities and make it easier to attack the system.

Third, emerging concepts in advanced aspect-oriented program design promise to allow for a separation of component, component-integration, security mechanism, and other concerns that, in turn, can enable important capabilities for the dependable design and dynamic adaptation of WSNs. Dynamic integration and re-configurability and the separate modularization of dynamically interchangeable security mechanisms and policy implementations are keys to enabling effective defense and self-repair. In particular, aspect-oriented separation of security-related code could facilitate the writing, verification, and updating of security code over time, allowing security countermeasures to evolve in the face of evolving threats.

Fourth, multi-hop wireless download can now support a real-time adaptive change to WSNs software. This can create diversity (downloading new components) and repair (downloading new allowable integration information), two concepts useful for addressing security attacks. Dynamically downloading and integrating updated security components is a key enabler for dynamic, evolving self-defense and repair under security attacks.

Milestones

1. Define new attack models for WSNs.
2. Create instances of lightweight security solutions for routing, localization, group sensor fusion, etc.
3. Develop hardware support for all aspects of security in WSNs.
4. Define a self-healing, security friendly architecture.
5. Implement and evaluate solutions in complete systems in realistic environments.

Summary

Wireless sensor networks have many uses for the military, e.g., they can provide surveillance in hostile areas, can help protect military installations, and can support urban warfare activities. The potential is unbounded. All of these capabilities will be jeopardized without built-in dependability and security capabilities. There is no way to finally solve the security problem as new attacks will always be conceived. We require solutions to enable WSNs to (i) operate in the presence of attacks, and (ii) evolve to support security changes needed over the lifetime of a system. The potential solutions mentioned here can help move WSNs from benign applications and environments to rugged and realistic systems that can achieve their true potential.

Some Issues in WSN, MANET and Cellular Security
(Position Paper)

Gene Tsudik[1]

ABSTRACT

In this position paper, we address some current limitations and challenges as well as emerging directions in three related areas of secure communication: (1) security in Wireless Sensor Networks – WSNs, (2) security in Mobile Ad Hoc Networks – MANETs, and, (3) security in Cellular Phone Networks.

WSN Security

Survivability and Intrusion Resilience: Sensors that obtain information by sensing the environment might not be able to propagate in real time. One basic reason is that, a batch of deployed sensors could form not a network *per se*, but, a collection of devices each responsible for its own sensed data. Some sensor settings do not involve sensors ”talking” to each other; instead, a sensor waits for a mobile collector/sink to pass by in order to off-load sensed data. Such an environment obviates certain network security issues but opens others. Notably, a sensor needs to minimize the amount of overall collected information while preserving its security. At the same time, it needs to cope with the risk of compromise. Neither issue comes up in typical WSNs considered in the literature. We envisage a need for new techniques that combine the cryptographic features of cryptographic *forward security* with *aggregation* (of MACs and signatures) in order to satisfy security requirements of such ”disconnected” sensor networks.

Secure Initialization: sensors are typically mass-produced and deployed simultaneously in batches. Unlike personal ubiquitous devices (such as cell-phones), sensors are not usually ”personal” and lack traditional means of input and output. (In particular, since a sensor is not a *computer*, in a traditional sense, it lacks an HCI.) A collection of sensors that needs to be deployed may need to be initialized to share a common secret key. Much work has been done in developing a plethora of key (pre-)distribution techniques, based on both public key and/or conventional cryptography. However, all such techniques are inapplicable in scenarios where sensors are not obtained in well-defined groups that can be initialized by the manufacturer.
If no secret keys are pre-distributed, security initialization must be done in an ad hoc fashion. It cannot be done via some wireless broadcast medium since doing so would be subject to trivial eavesdropping. Doing it with wires or other direct physical connection is awkward and unscalable. Consequently, new techniques are needed that address both security and scalability. One recent proposal called ”Shake-Them-Up” addresses the security issue to an extent, however, scalability remains to be tackled.

MANET Security

Anonymous Routing: we consider hostile MANET scenarios where network topology undergoes constant changes and current topology represents sensitive information which must be kept confidential even from MANET nodes themselves. (Troops on the battlefield is one prominent example.) In such cases, existing routing protocols are unsuitable and new packet forwarding methods must be developed.

Location-Based Addressing: in a MANET environments where nodes are mutually suspicious (e.g., because capture/compromise are possible) addressing and packet forwarding based on long-term identities is unsafe. This is because identities tend to reveal current locations of nodes and allow tracking of nodes as the network topology changes. At the same addressing based on location only (without knowing whether anyone is there) is not optimal since a picked location might be empty and effort expended in discovering this is essentially wasted. Thus, it makes more sense to periodically announce each node’s location, thus making it possible to use location as a reliable current address of the destination. We claim that, if a sufficient

[1] Department of Computer Science, University of California, Irvine. gts_AT_ics.uci.edu

fraction of all nodes change locations between successive updates, tracking of nodes becomes infeasible. Moreover, if nodes have a way to authenticate their routing updates in an anonymous and un-linkable fashion, the network becomes more secure than is possible with current secure MANET routing techniques. The biggest challenge is to come up with a MANET architecture that allows this kind of operation in an efficient manner.

Cellular/Mobile Phone Security

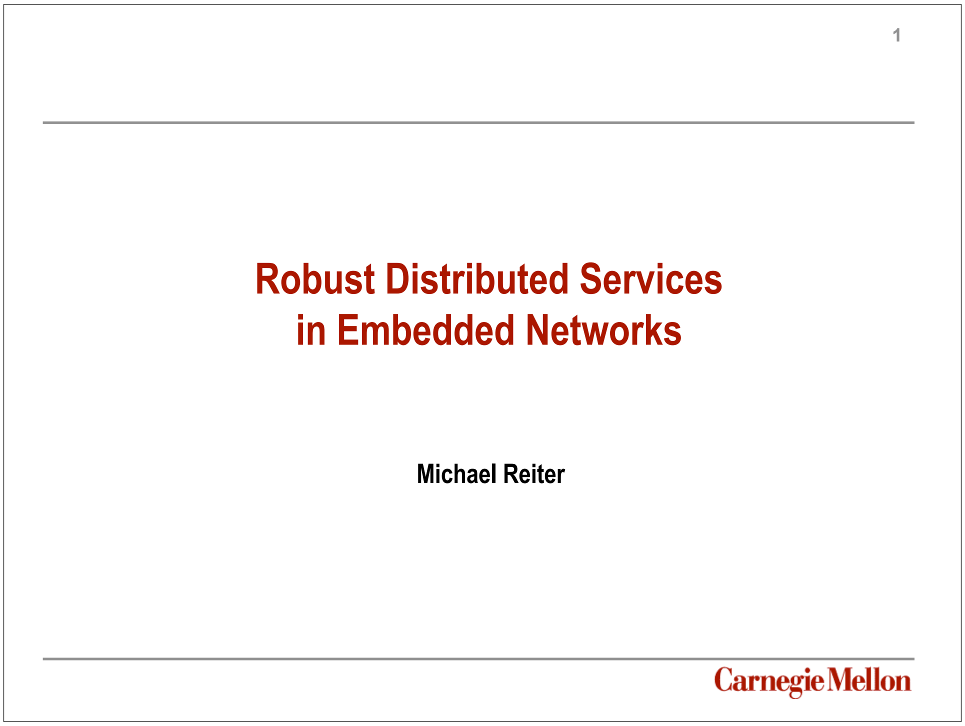
Security in cellular phone networks has been studied extensively since early 90-s and there is a large body of literature on the subject. However, two prominent problems remain, as we highlight below.

Secure Pairing: sometimes referred to as *Secure First Connect*, this issue has to do with establishing a secure means of communication between two devices, at least one of which is a cell phone. The problem is exacerbated by three factors:
- heterogeneous devices (both phones and others) varying widely in terms of features (means of input/output)
- lack of any standard security infrastructure such as a common PKI
- inability to rely solely upon human-imperceptible means of communication due to man-in-the-middle attacks
- consequent reliance on the human user, which requires minimizing user burden while offering sufficient security

Notable secure pairing techniques proposed to-date involve using so-called location-limited side-channels. Each requires some direct involvement of the human user but they differ in the type and degree thereof. It has been widely accepted that involving the human user is unavoidable. At the same time, no technique is universal, even when it comes to pairing two similar cellphones. On the one hand, the design space of possible secure pairing techniques has not been thoroughly explored. (Methods like ”Resurrecting Duckling”, ”Talking-to-Strangers”, ”Seeing-is-Believing”, ”Loud-and-Clear” and ”HAPADEP” notwithstanding.) Moreover, usability studies have been undertaken only recently and much more work is remains to be done to adequately assess usability factors of already proposed techniques.

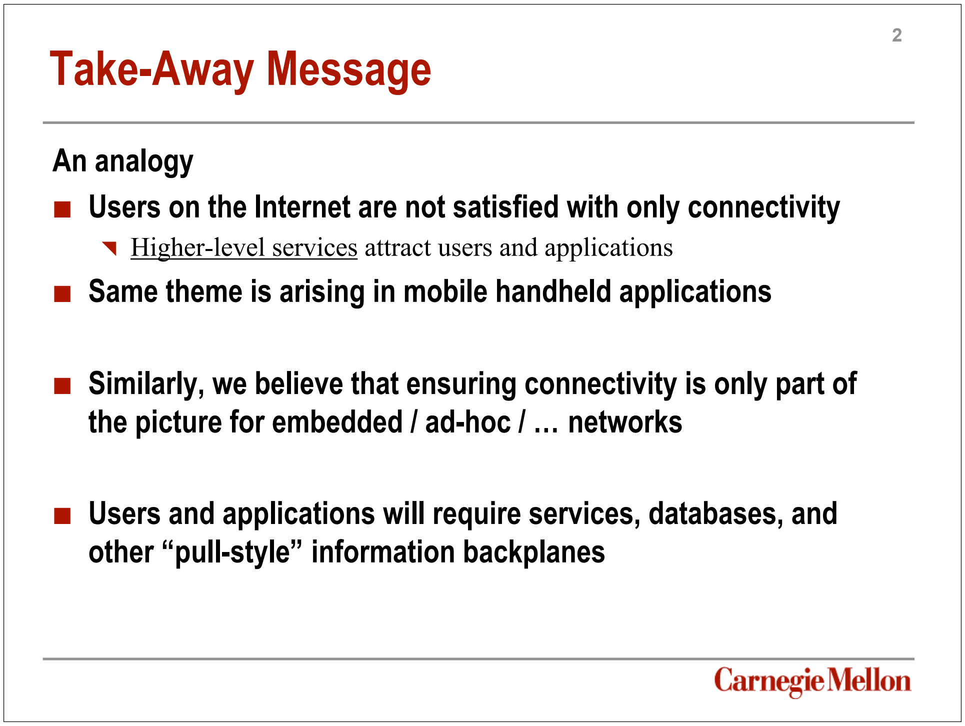
Anonymous Roaming: This issue refers to the ability to use one’s cellular phone without exposing the phone’s long-term identifier (e.g. IMSI in GSM) to the roaming network/provider. While most, if not all, cellular networks in operation today require the notion of ”home” for each subscriber (SIM or phone unit, depending on the underlying standard), there is no inherent need to disclose the long-term identifier (There is, however, a legitimate need to disclose the ”home” provider, however, that is a far cry from disclosing the actual phone identifier.) The need for roaming anonymously has been recognized for quite some time. However, despite the fact that the technology (protocols, designs, cryptographic primitives) is readily available, anonymous roaming is not available on any current cellular network.

The research challenge in anonymous roaming is not great; it boils down to coming up with concrete set of secure cryptographic protocols that support roaming anonymity and convincing the providers as well as manufacturers to offer anonymity as a service.



Robust Distributed Services in Embedded Networks

Michael Reiter



Take-Away Message

An analogy

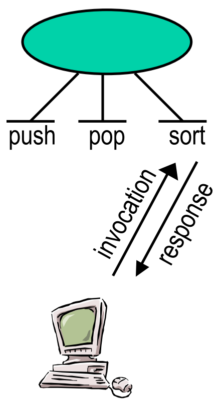
- Users on the Internet are not satisfied with only connectivity
  - Higher-level services attract users and applications
- Same theme is arising in mobile handheld applications
- Similarly, we believe that ensuring connectivity is only part of the picture for embedded / ad-hoc / … networks
- Users and applications will require services, databases, and other "pull-style" information backplanes

What Makes This Difficult?

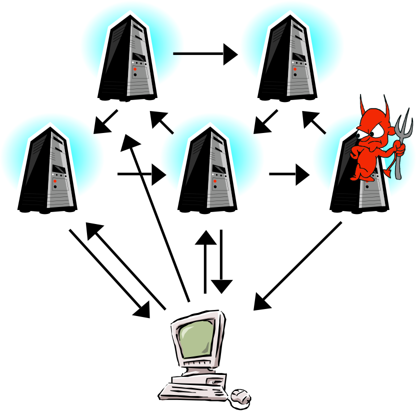
- If your embedded / ad-hoc network is autonomous, it may have no servers!
  - At least not in the typical sense of that word
- A server is typically
  - Well provisioned and maintained
  - Reliably connected
  - Relatively trustworthy
- Embedded / ad hoc networks may lack any such nodes

Survivable Distributed Services

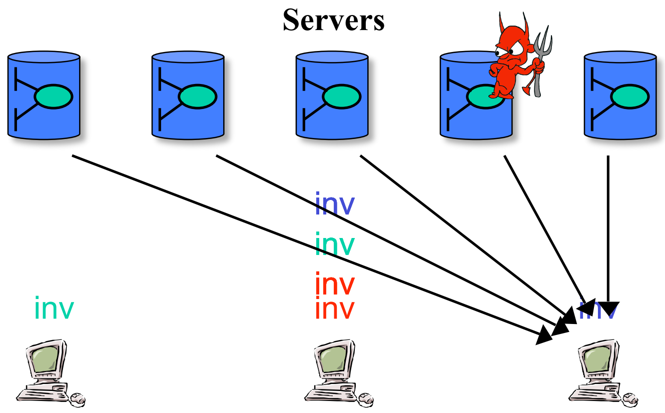
- Service, or object, abstraction



- Implementation



Traditional Approach: State Machine Replication

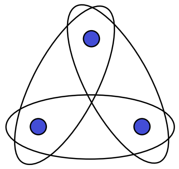


- Offers no load dispersion, and degrades as system scales

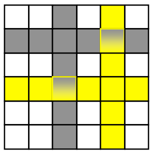
Quorum Systems

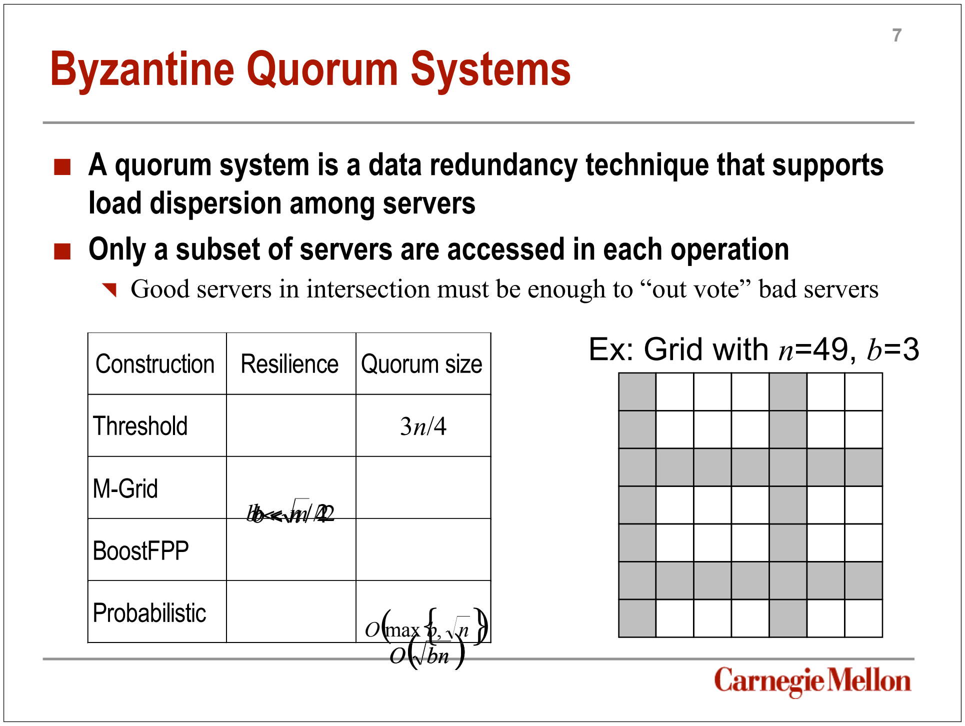
- **Quorum systems:**
  - Basic tool for synchronization in distributed systems
  - A set of subsets (*quorums*) of a universe U of logical elements, having intersection property (any pair of quorums intersect)

Majority



Grid





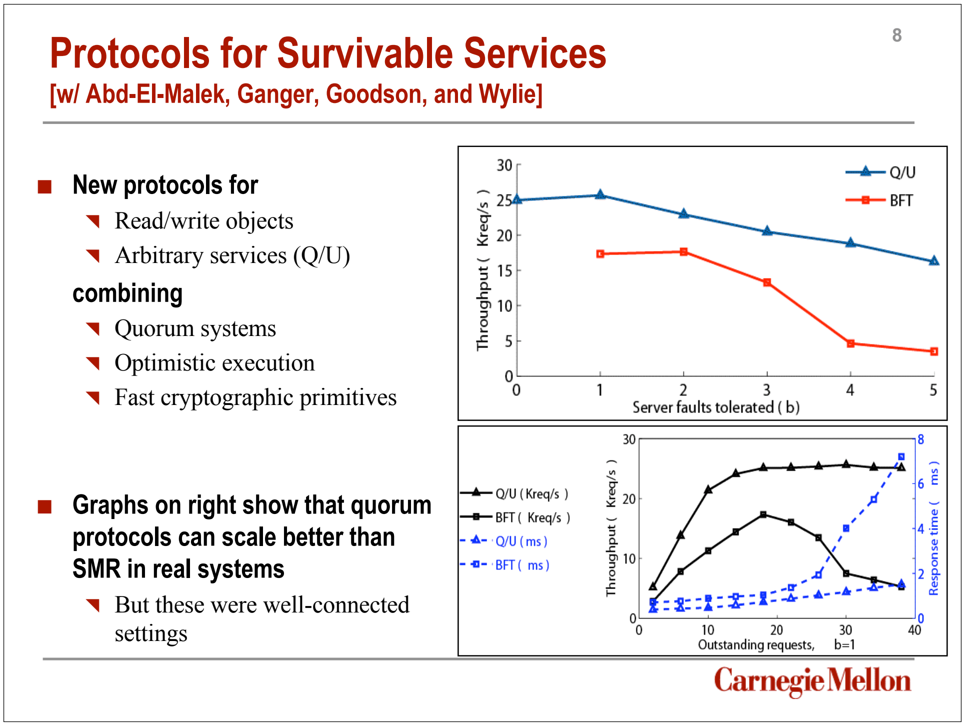
Byzantine Quorum Systems

- A quorum system is a data redundancy technique that supports load dispersion among servers
- Only a subset of servers are accessed in each operation
  - Good servers in intersection must be enough to “out vote” bad servers

Construction	Resilience	Quorum size
Threshold		$3n/4$
M-Grid	$b < \sqrt{n}/2$ $b < n/4$	
BoostFPP		
Probabilistic		$O(\max\{b, \sqrt{n}\})$ $O(\sqrt{bn})$

Ex: Grid with n=49, b=3

Carnegie Mellon



Protocols for Survivable Services

[w/ Abd-El-Malek, Ganger, Goodson, and Wylie]

- New protocols for
  - Read/write objects
  - Arbitrary services (Q/U)

combining
  - Quorum systems
  - Optimistic execution
  - Fast cryptographic primitives

- Graphs on right show that quorum protocols can scale better than SMR in real systems
  - But these were well-connected settings

Q/U
BFT
Throughput (Kreq/s)
Server faults tolerated (b)
0
1
2
3
4
5
10
15
20
25
30
Q/U (Kreq/s)
BFT (Kreq/s)
Q/U (ms)
BFT (ms)
Throughput (Kreq/s)
Response time (ms)
Outstanding requests, b=1
40
6
8

Carnegie Mellon

9

Dealing with Network Effects

- Network effects are likely to be just as important in embedded / ad hoc networks as load dispersion

- Even worse, minimizing network delays for accessing quorums can be in conflict with load dispersion
  - May have to bypass a close but heavily-loaded quorum in favor of a less-loaded but more distant quorum

- Can we balance this tradeoff?

Carnegie Mellon

10

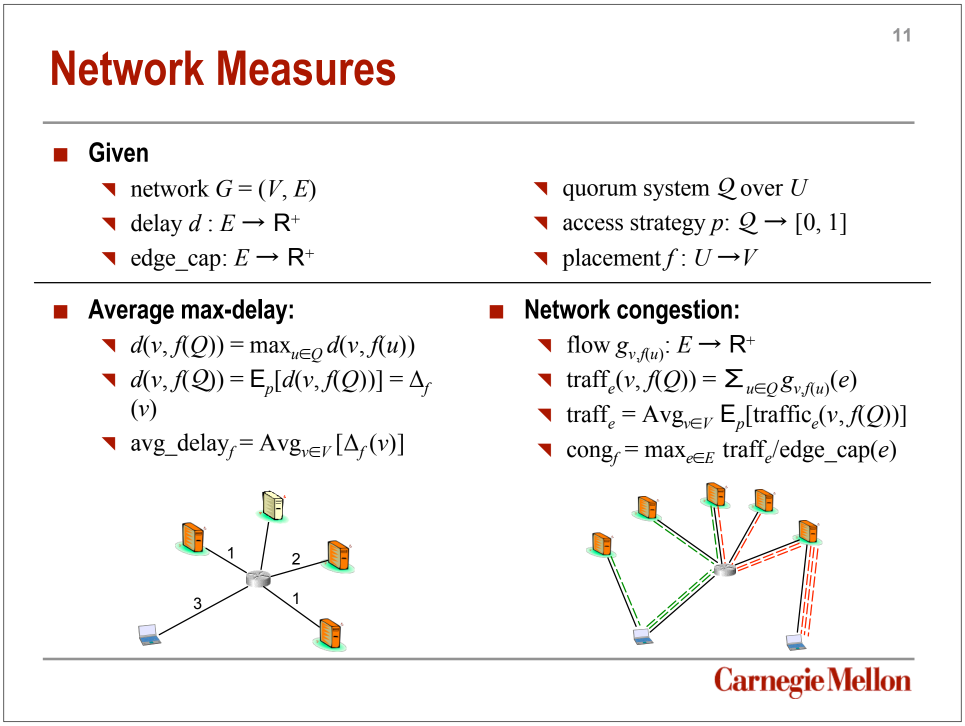
Quorum Placement Problems

- Place "good" quorum systems on network
  - to minimize network-specific measures
  - preserve "goodness"

- Goodness = load
  - Assume each quorum Q is accessed with probability $p(Q)$
  - $load_p(u) = \Sigma_{Q:\, u \in Q}\, p(Q)$

- Network measures:
  - Average delay observed by clients when accessing quorum system
  - Network congestion induced by clients accessing quorum system

Carnegie Mellon

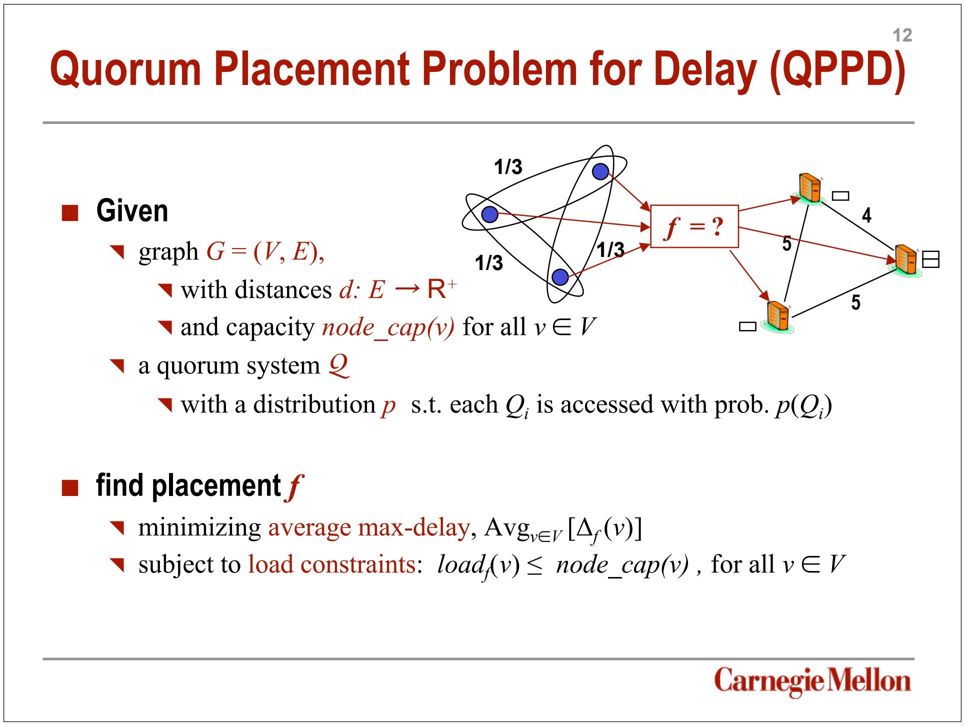


11

Network Measures

- **Given**
  - network $G = (V, E)$
  - delay $d : E \to \mathrm{R}^+$
  - edge_cap: $E \to \mathrm{R}^+$
  - quorum system Q over U
  - access strategy p: $Q \to [0, 1]$
  - placement $f : U \to V$
- **Average max-delay:**
  - $d(v, f(Q)) = \max_{u \in Q} d(v, f(u))$
  - $d(v, f(Q)) = \mathrm{E}_p[d(v, f(Q))] = \Delta_f(v)$
  - avg_delay$_f$ = $\mathrm{Avg}_{v \in V} [\Delta_f(v)]$
- **Network congestion:**
  - flow $g_{v,f(u)}$: $E \to \mathrm{R}^+$
  - traff$_e(v, f(Q)) = \Sigma_{u \in Q}\, g_{v,f(u)}(e)$
  - traff$_e$ = $\mathrm{Avg}_{v \in V}\ \mathrm{E}_p[\text{traffic}_e(v, f(Q))]$
  - cong$_f$ = $\max_{e \in E}$ traff$_e$/edge_cap(e)

1
2
3
1

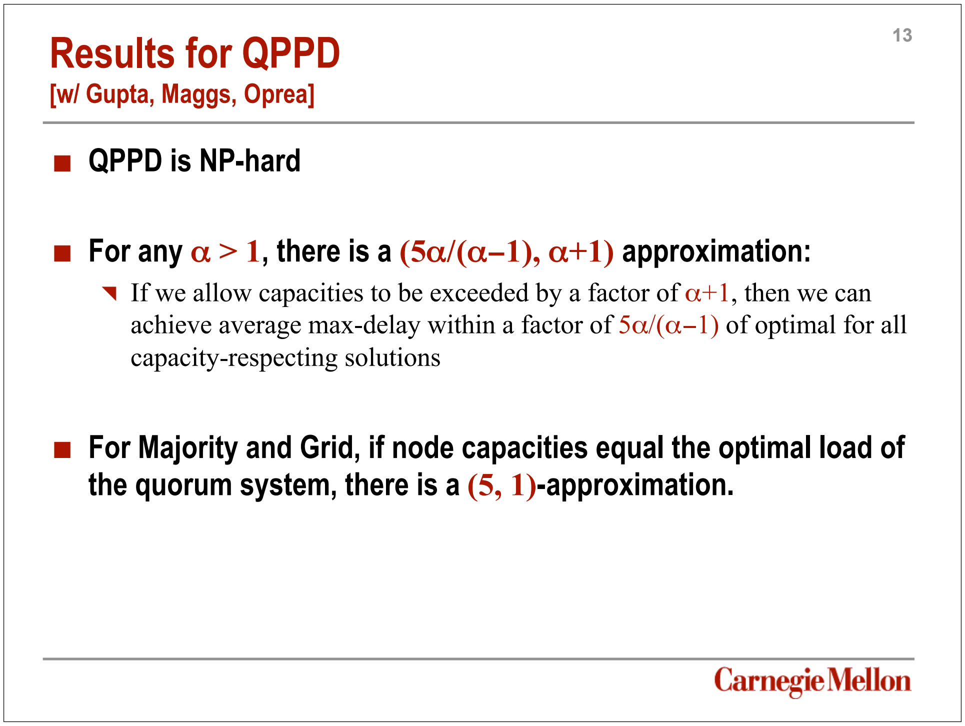


12

Quorum Placement Problem for Delay (QPPD)

- **Given**
  - graph $G = (V, E)$,
    - with distances d: $E \to \mathrm{R}^+$
    - and capacity $node\_cap(v)$ for all $v \in V$
  - a quorum system Q
    - with a distribution p s.t. each Q_i is accessed with prob. $p(Q_i)$
- **find placement f**
  - minimizing average max-delay, $\mathrm{Avg}_{v \in V} [\Delta_f(v)]$
  - subject to load constraints: $load_f(v) \leq node\_cap(v)$, for all $v \in V$

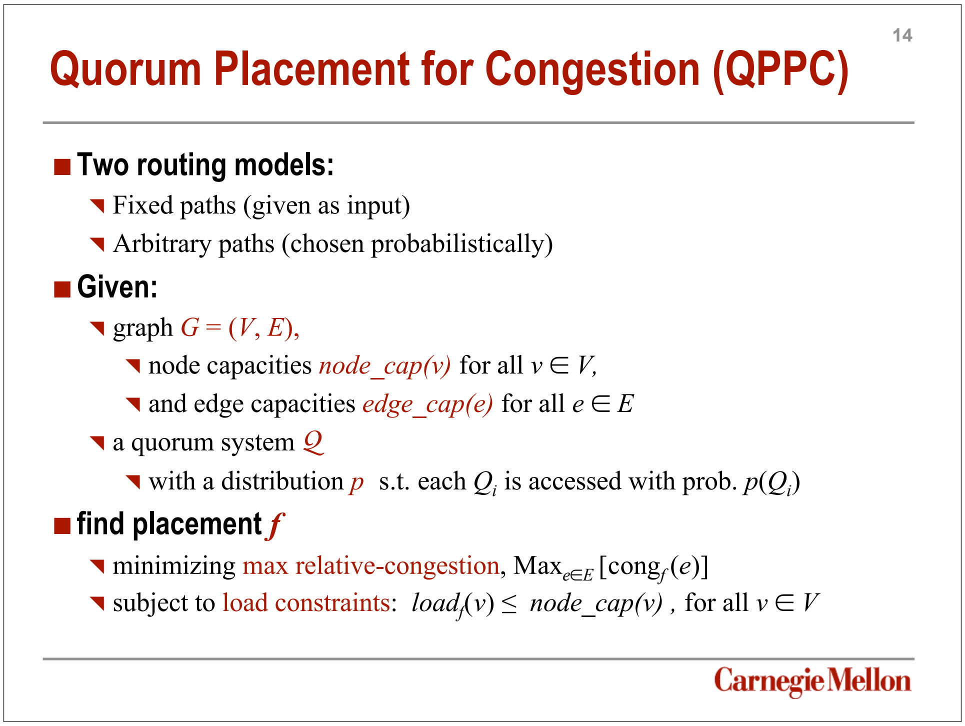
1/3
1/3
1/3
f = ?
4
5
5



Results for QPPD

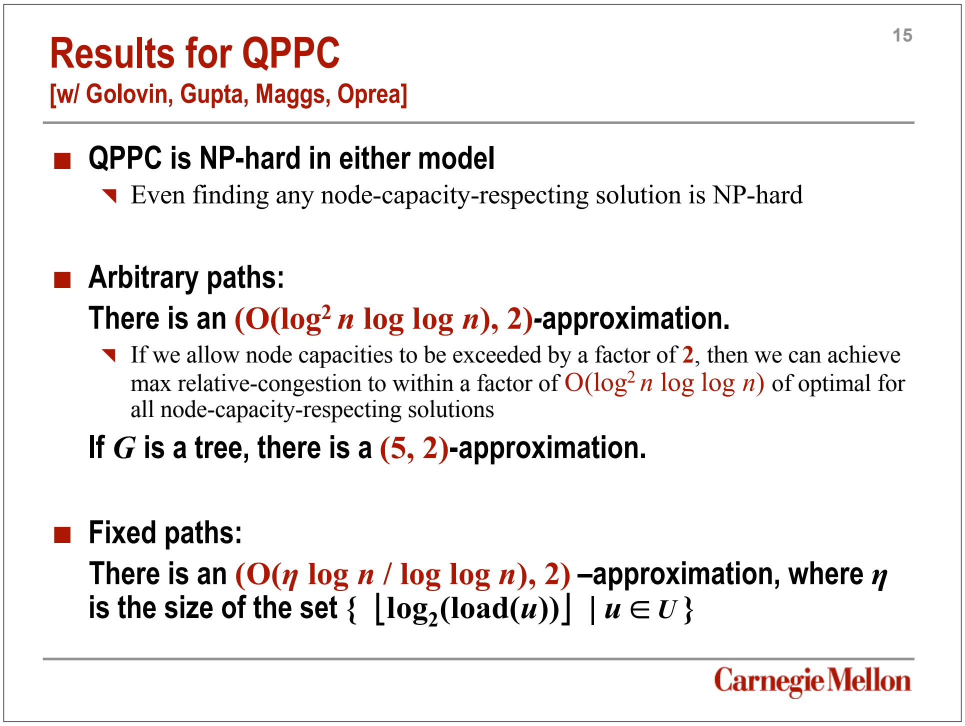
[w/ Gupta, Maggs, Oprea]

- **QPPD is NP-hard**
- **For any $\alpha > 1$, there is a $(5\alpha/(\alpha-1), \alpha+1)$ approximation:**
  - If we allow capacities to be exceeded by a factor of $\alpha+1$, then we can achieve average max-delay within a factor of $5\alpha/(\alpha-1)$ of optimal for all capacity-respecting solutions
- **For Majority and Grid, if node capacities equal the optimal load of the quorum system, there is a $(5, 1)$-approximation.**



Quorum Placement for Congestion (QPPC)

- **Two routing models:**
  - Fixed paths (given as input)
  - Arbitrary paths (chosen probabilistically)
- **Given:**
  - graph $G = (V, E)$,
    - node capacities $node\_cap(v)$ for all $v \in V$,
    - and edge capacities $edge\_cap(e)$ for all $e \in E$
  - a quorum system $\mathcal{Q}$
    - with a distribution p s.t. each Q_i is accessed with prob. $p(Q_i)$
- **find placement f**
  - minimizing max relative-congestion, $\mathrm{Max}_{e \in E}\,[\mathrm{cong}_f(e)]$
  - subject to load constraints: $load_f(v) \leq node\_cap(v)$, for all $v \in V$



15

Results for QPPC

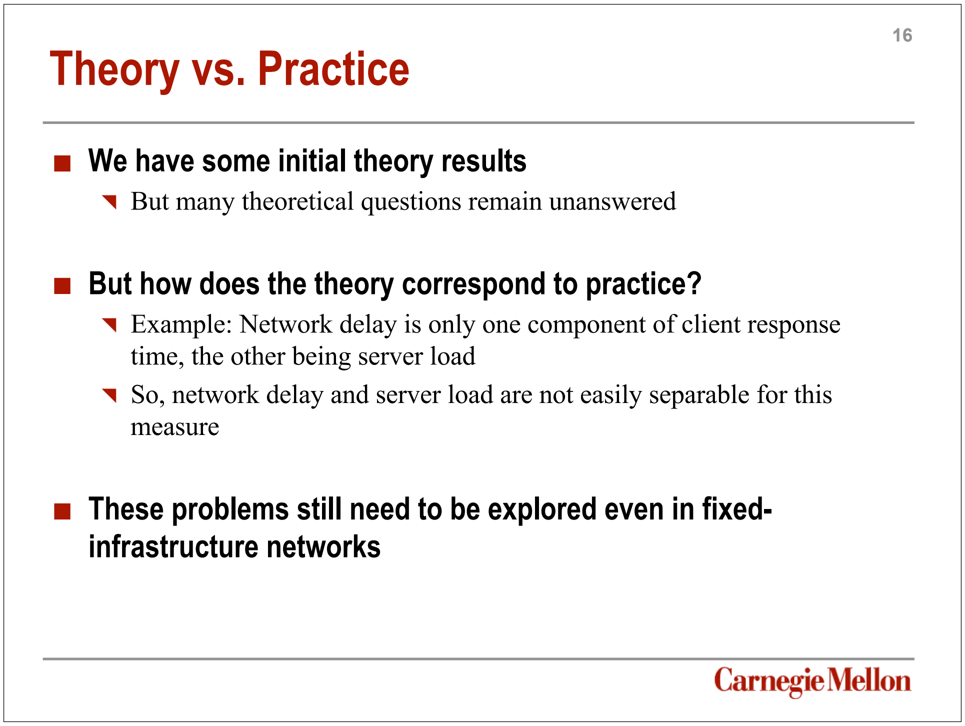
[w/ Golovin, Gupta, Maggs, Oprea]

- **QPPC is NP-hard in either model**
  - Even finding any node-capacity-respecting solution is NP-hard
- **Arbitrary paths:**

  There is an $(O(\log^2 n \log\log n), 2)$-approximation.
  - If we allow node capacities to be exceeded by a factor of **2**, then we can achieve max relative-congestion to within a factor of $O(\log^2 n \log\log n)$ of optimal for all node-capacity-respecting solutions

  If G is a tree, there is a $(5, 2)$-approximation.
- **Fixed paths:**

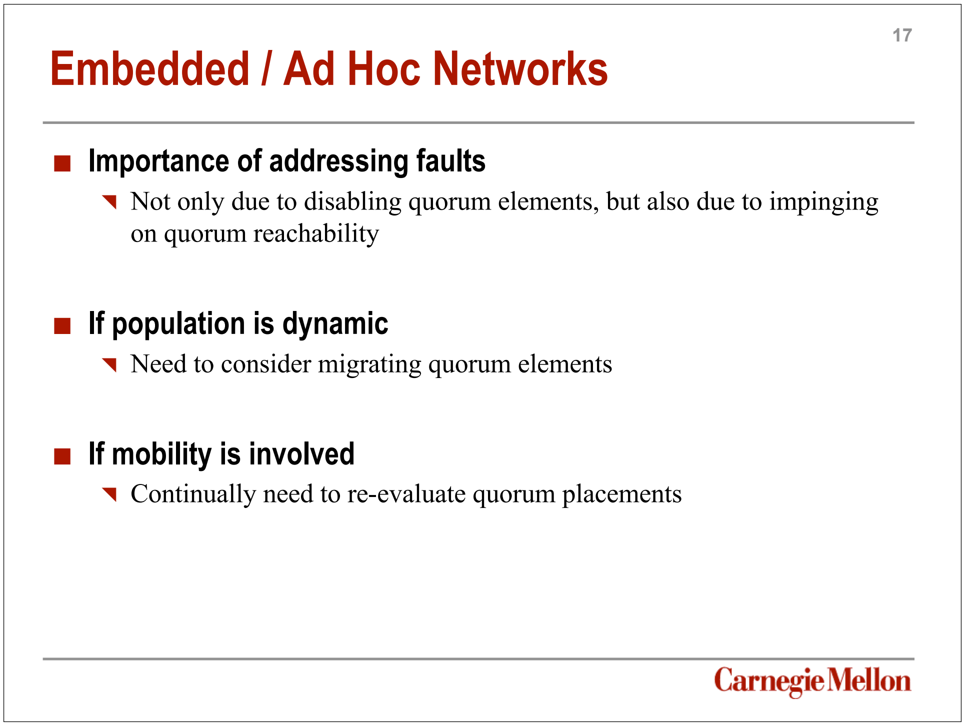
  There is an $(O(\eta \log n / \log\log n), 2)$ –approximation, where η is the size of the set $\{ \lfloor \log_2(\text{load}(u)) \rfloor \mid u \in U \}$



16

Theory vs. Practice

- **We have some initial theory results**
  - But many theoretical questions remain unanswered
- **But how does the theory correspond to practice?**
  - Example: Network delay is only one component of client response time, the other being server load
  - So, network delay and server load are not easily separable for this measure
- **These problems still need to be explored even in fixed-infrastructure networks**



Embedded / Ad Hoc Networks

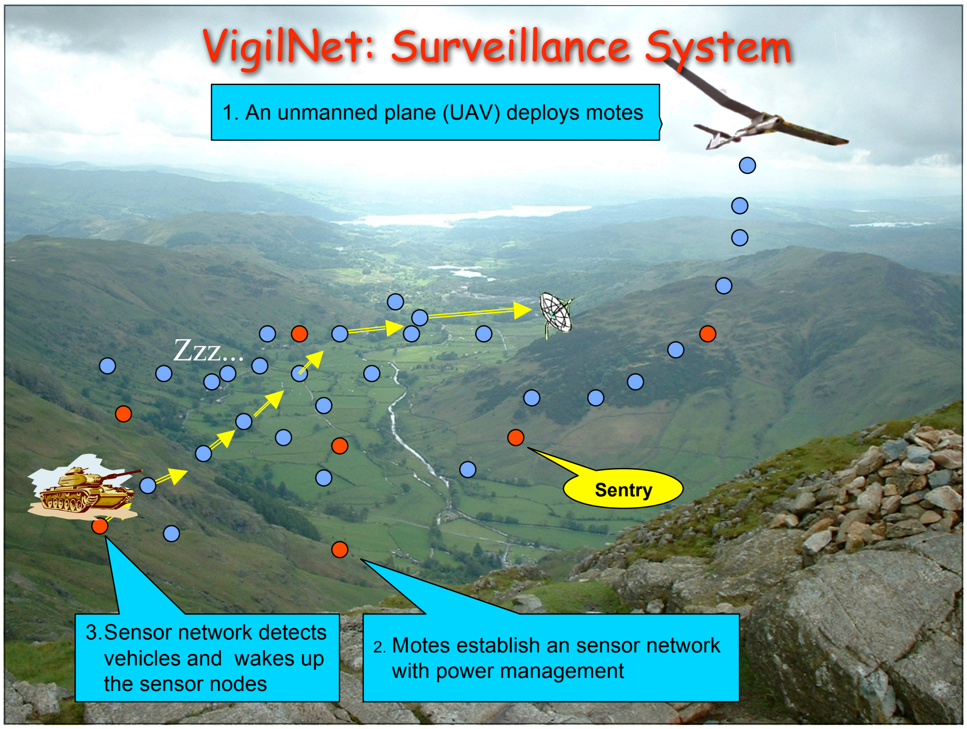
- **Importance of addressing faults**
  - Not only due to disabling quorum elements, but also due to impinging on quorum reachability
- **If population is dynamic**
  - Need to consider migrating quorum elements
- **If mobility is involved**
  - Continually need to re-evaluate quorum placements

Adaptable and Reactive Security for Wireless Sensor Networks

John A. Stankovic
Department of Computer Science
University of Virginia

Outline

- Brief Motivation
- Adaptable Self-Healing Architecture
  - Components
  - AOP
  - Robust Decentralized Control
  - Lightweight Security Components
- Systems of Systems
- Summary

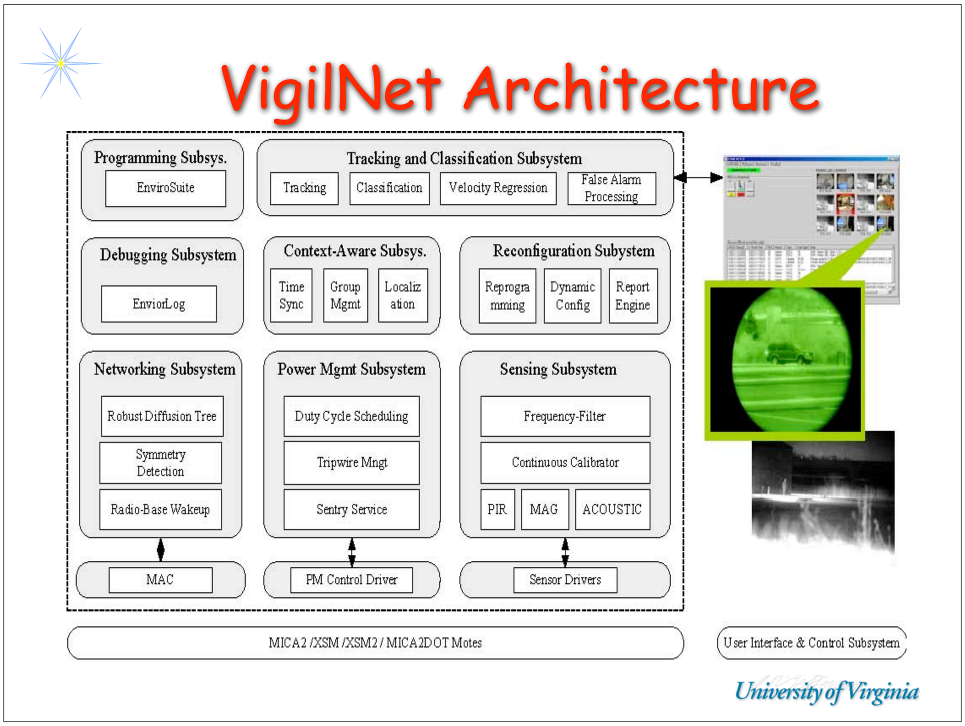


VigilNet: Surveillance System

1. An unmanned plane (UAV) deploys motes

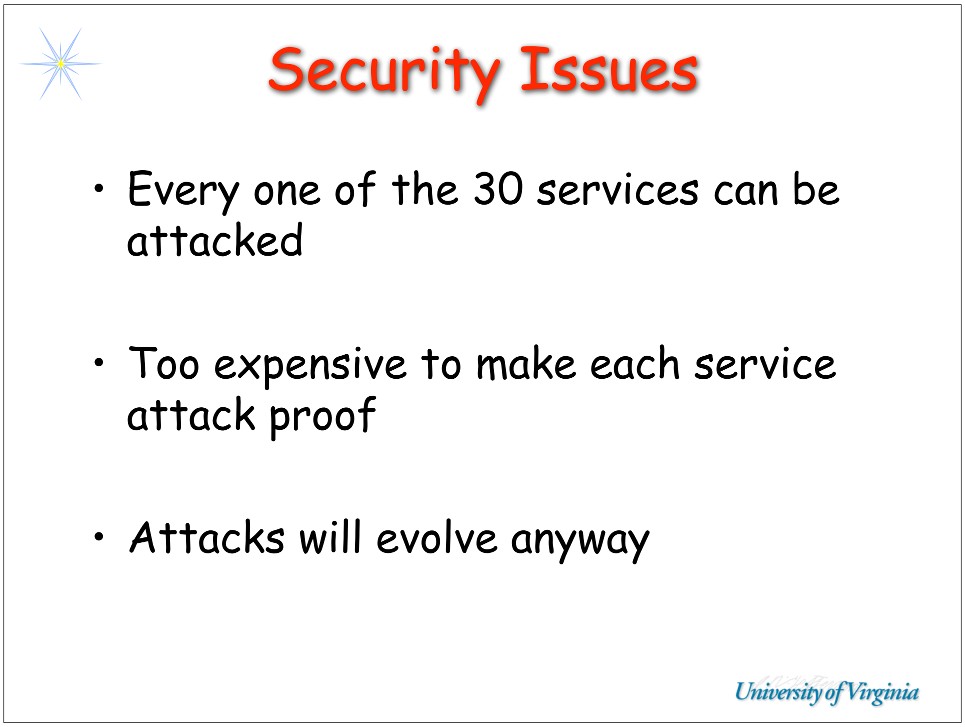
2. Motes establish an sensor network with power management

3.Sensor network detects vehicles and wakes up the sensor nodes



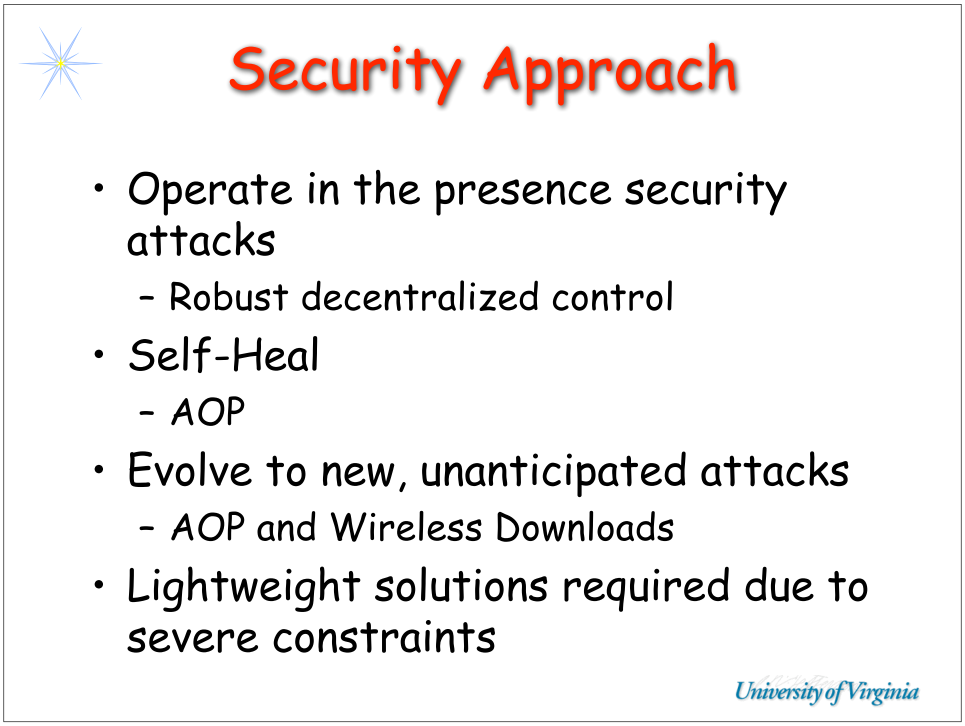
VigilNet Architecture

University of Virginia



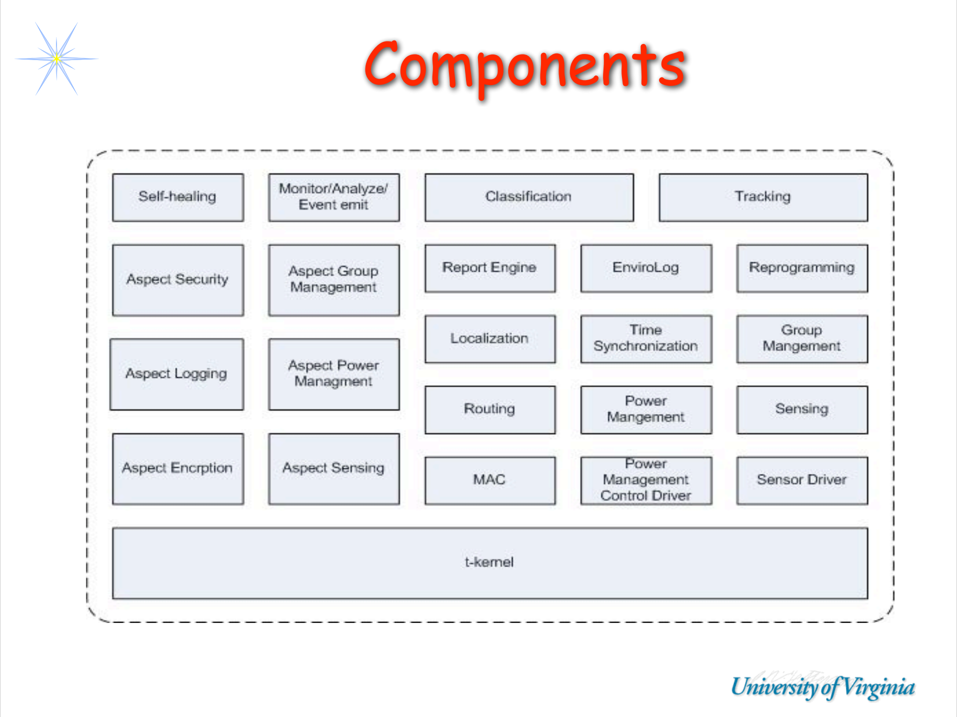
Security Issues

- Every one of the 30 services can be attacked
- Too expensive to make each service attack proof
- Attacks will evolve anyway



Security Approach

- Operate in the presence security attacks
  - Robust decentralized control
- Self-Heal
  - AOP
- Evolve to new, unanticipated attacks
  - AOP and Wireless Downloads
- Lightweight solutions required due to severe constraints



Components

Self-healing

Monitor/Analyze/ Event emit

Classification

Tracking

Aspect Security

Aspect Group Management

Report Engine

EnviroLog

Reprogramming

Localization

Time Synchronization

Group Mangement

Aspect Logging

Aspect Power Managment

Routing

Power Mangement

Sensing

Aspect Encrption

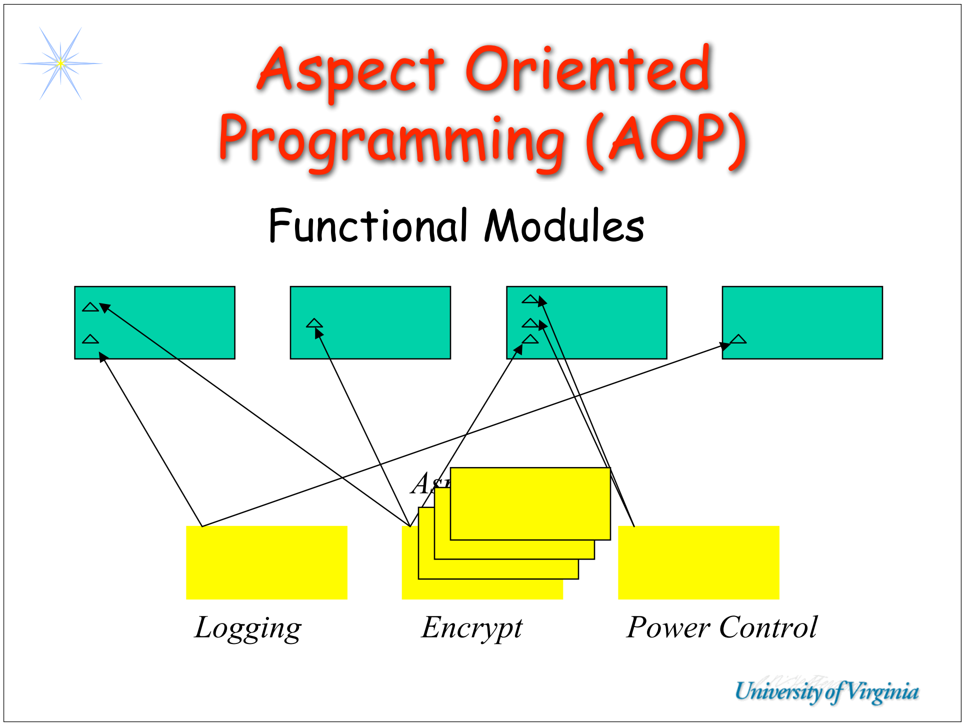
Aspect Sensing

MAC

Power Management Control Driver

Sensor Driver

t-kernel



Aspect Oriented Programming (AOP)

Functional Modules

Logging

Encrypt

Power Control

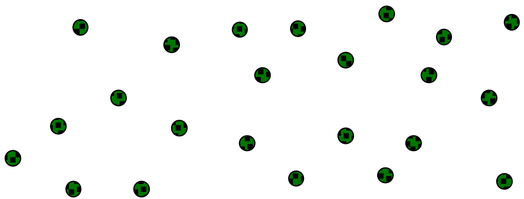
Unanticipated Attacks

- What if advice was not available on the nodes
  - Typical for an unanticipated attack

  - Report event to base station
  - Find/Write new aspects
  - Disseminate to nodes

Decentralized Control

- Large Numbers of Nodes
  - Aggregate Behavior Emerges
  - Control/Guarantee Behavior



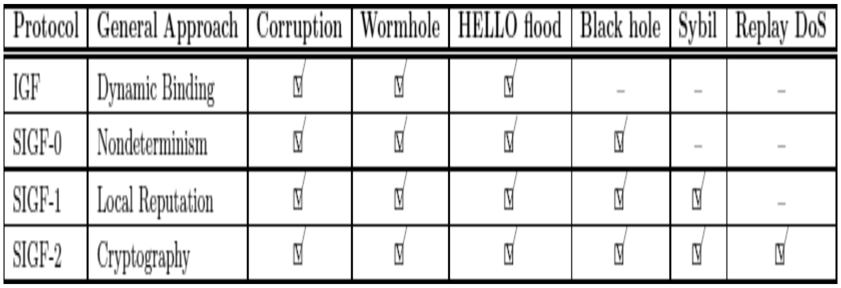
- Redundancy
- Mask faults/ attacks
- Uniformity a problem/diversity

Lightweight Components

- Secure (reactive/adaptive) routing
- Localization

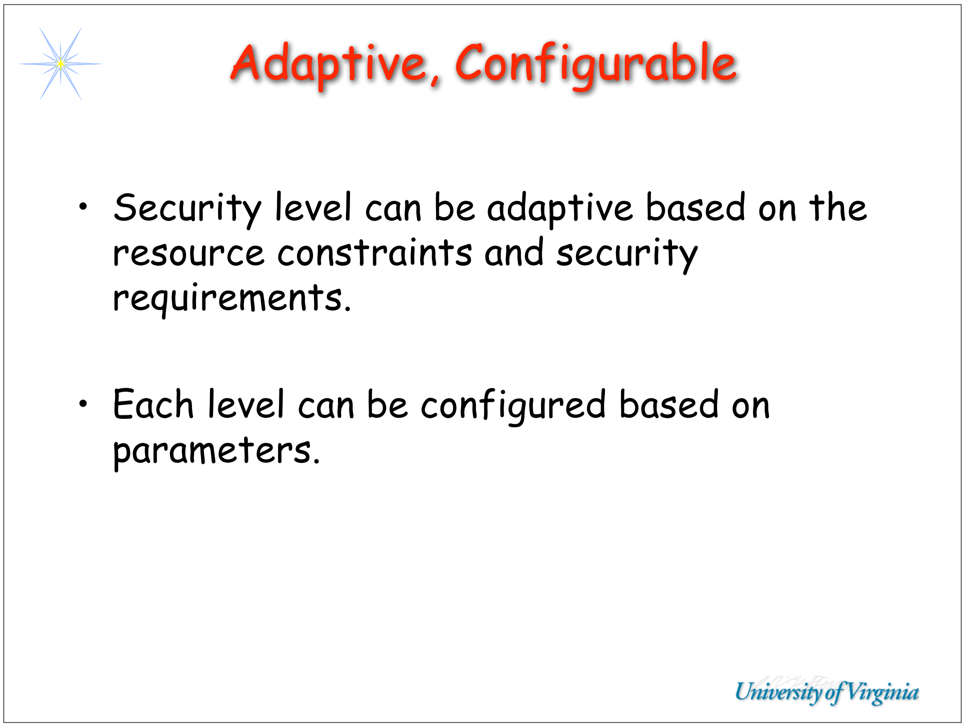
SIGF

- The SIGF family provides incremental steps between stateless and shared-state protocols.



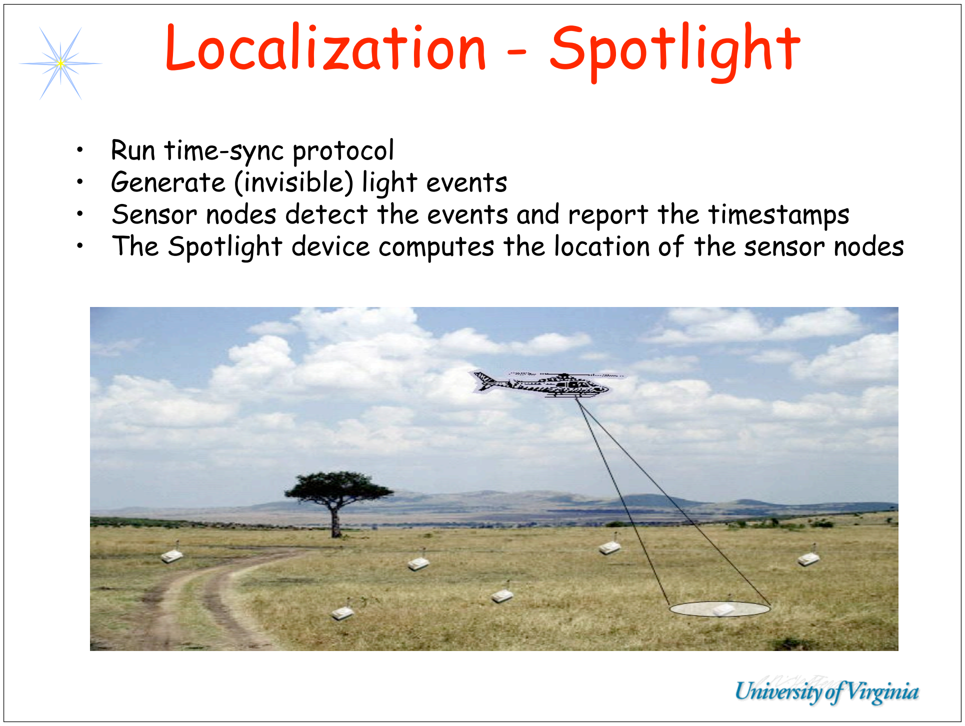
Protocol	General Approach	Corruption	Wormhole	HELLO flood	Black hole	Sybil	Replay DoS
IGF	Dynamic Binding	☑	☑	☑	–	–	–
SIGF-0	Nondeterminism	☑	☑	☑	☑	–	–
SIGF-1	Local Reputation	☑	☑	☑	☑	☑	–
SIGF-2	Cryptography	☑	☑	☑	☑	☑	☑

- SIGF allows efficient operation when no attacks are present, and good enough security when they are.



Adaptive, Configurable

- Security level can be adaptive based on the resource constraints and security requirements.
- Each level can be configured based on parameters.

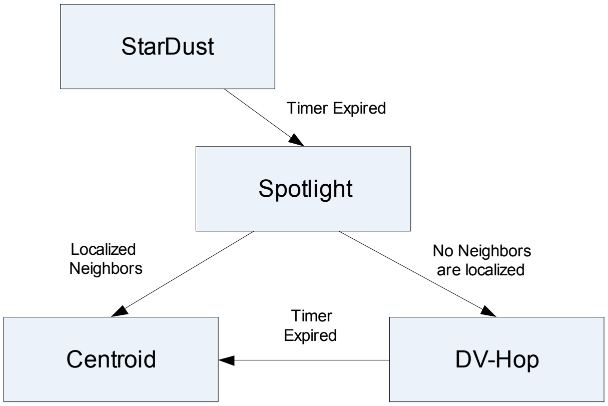


Localization - Spotlight

- Run time-sync protocol
- Generate (invisible) light events
- Sensor nodes detect the events and report the timestamps
- The Spotlight device computes the location of the sensor nodes

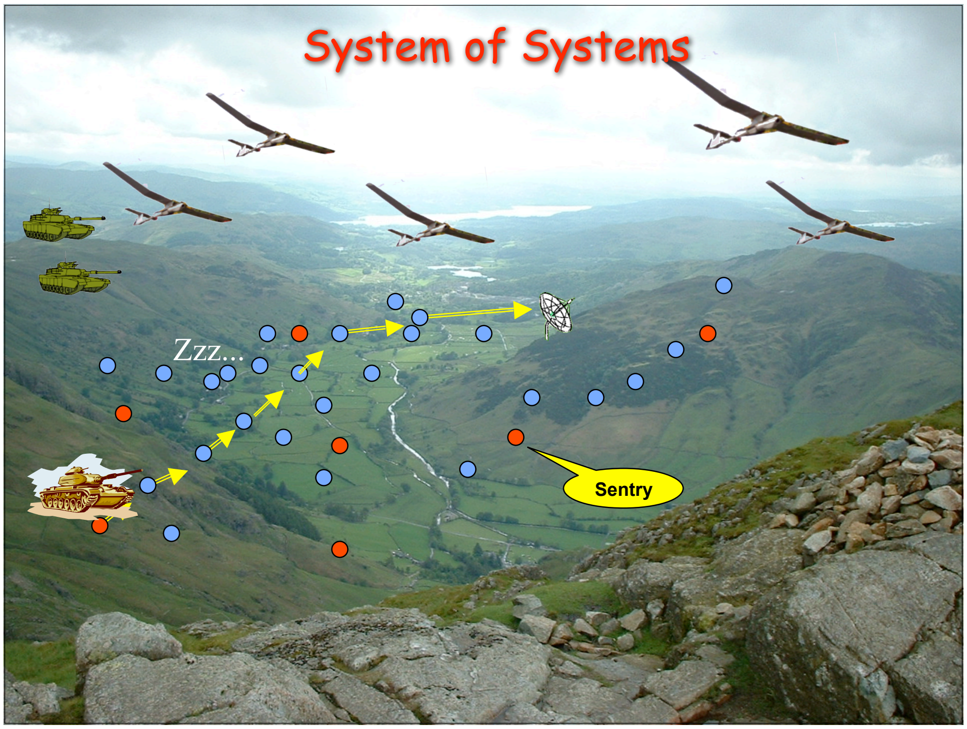
Localization Robustness

- Execute combination of protocols



University of Virginia

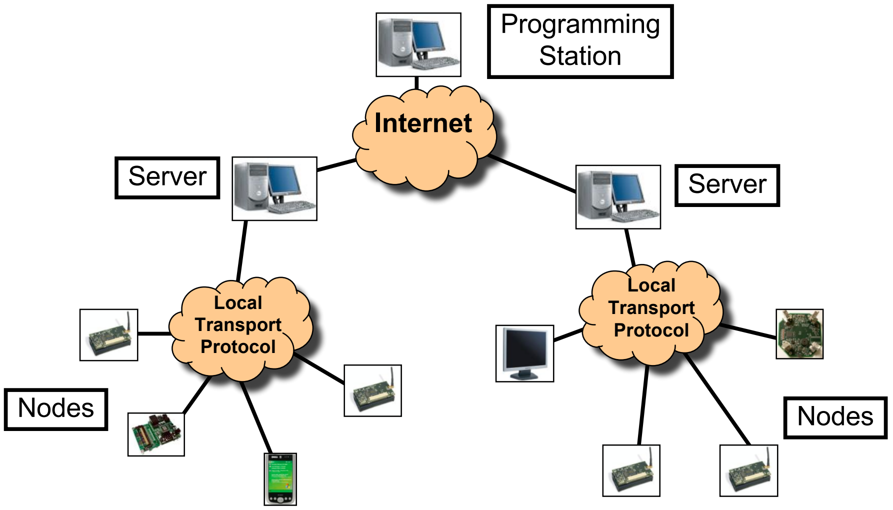
System of Systems



Systems of Systems

- Inter-system security
- How to program and debug to ensure
  - Behavior
  - Robustness

System Architecture



Programming Station

Internet

Server

Server

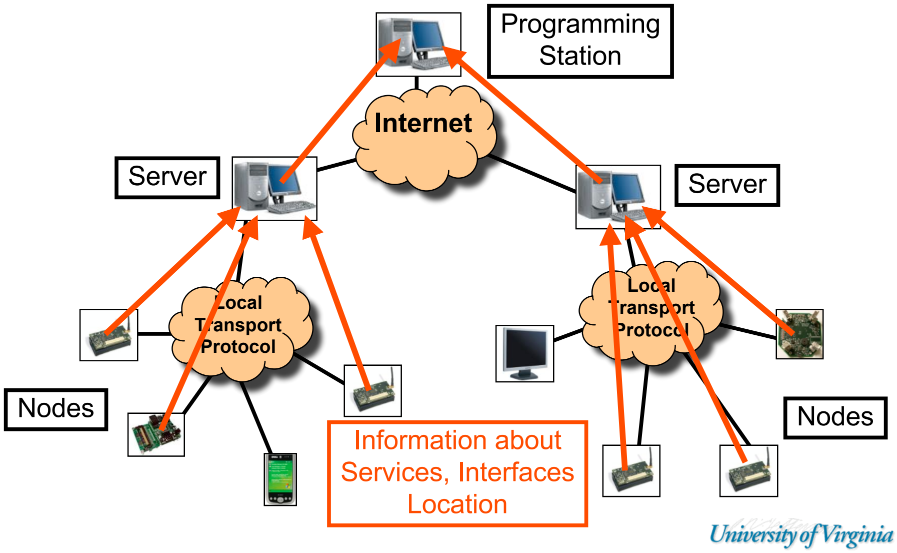
Local Transport Protocol

Local Transport Protocol

Nodes

Nodes

System Architecture



Programming Station

Internet

Server

Server

Local Transport Protocol

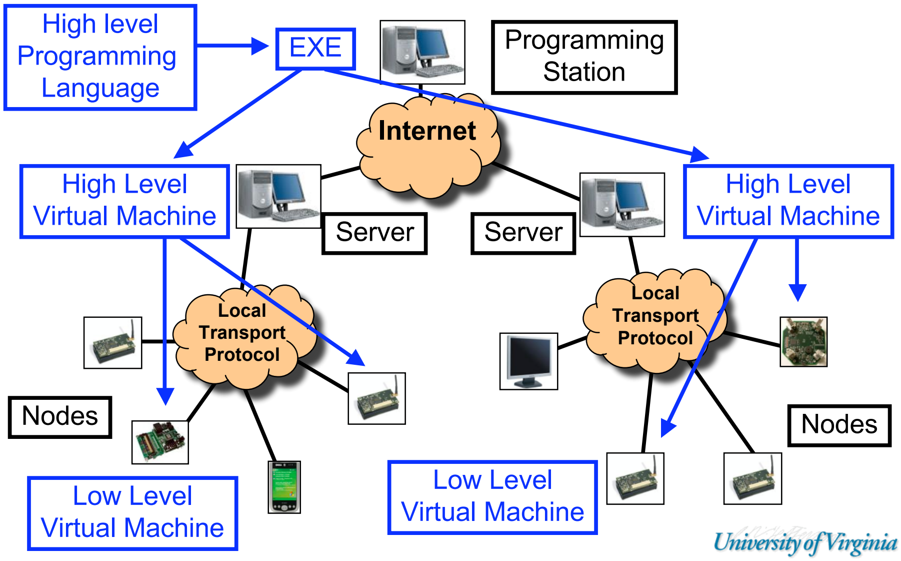
Local Transport Protocol

Nodes

Nodes

Information about Services, Interfaces Location

System Architecture



High level Programming Language

EXE

Programming Station

Internet

High Level Virtual Machine

High Level Virtual Machine

Server

Server

Local Transport Protocol

Local Transport Protocol

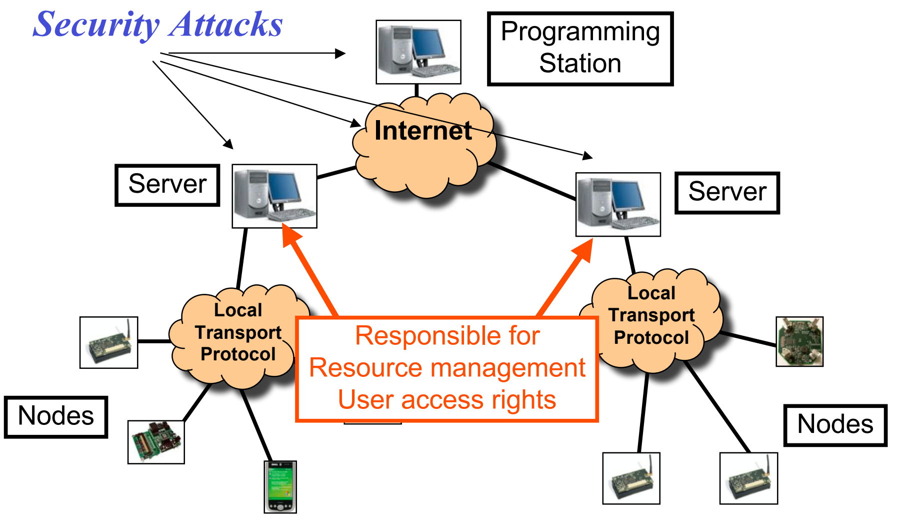
Nodes

Nodes

Low Level Virtual Machine

Low Level Virtual Machine

System Architecture



Security Attacks

Programming Station

Internet

Server

Server

Local Transport Protocol

Local Transport Protocol

Responsible for
Resource management
User access rights

Nodes

Nodes

Summary

- Security in WSN
  - Lightweight
  - Attack resilience
  - Evolve for unexpected attacks
- Security in Systems of Systems
- Self-Healing Architecture
  - AOP extensions
  - Adaptive and lightweight protocols

Acknowledgements

- Anthony Wood
- Hua Cao
- Radu Stoleru

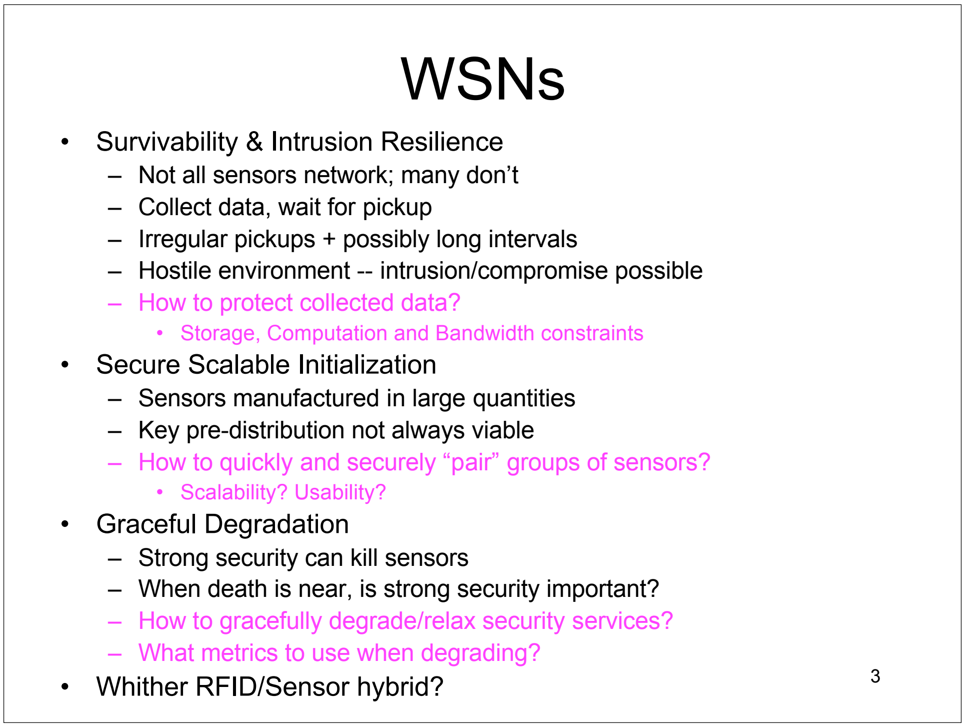
University of Virginia

Some Issues in MANET, Wireless & Cellular Security/Privacy

Gene Tsudik
UC Irvine
gts_AT_ics.uci.edu

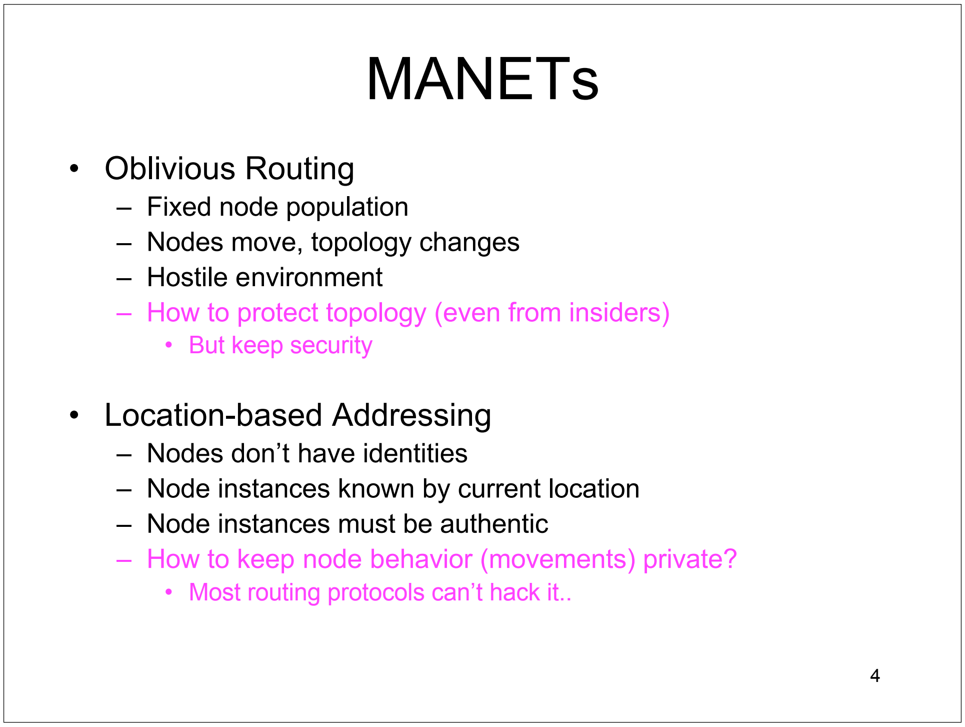
Outline

- WSNs:
  - Survivability & Intrusion Resilience
  - Secure Scalable Initialization
  - Graceful Degradation
- MANETs:
  - Oblivious Routing
  - Location-based Addressing
- Cellular:
  - Secure Association
  - Anonymous Mobility



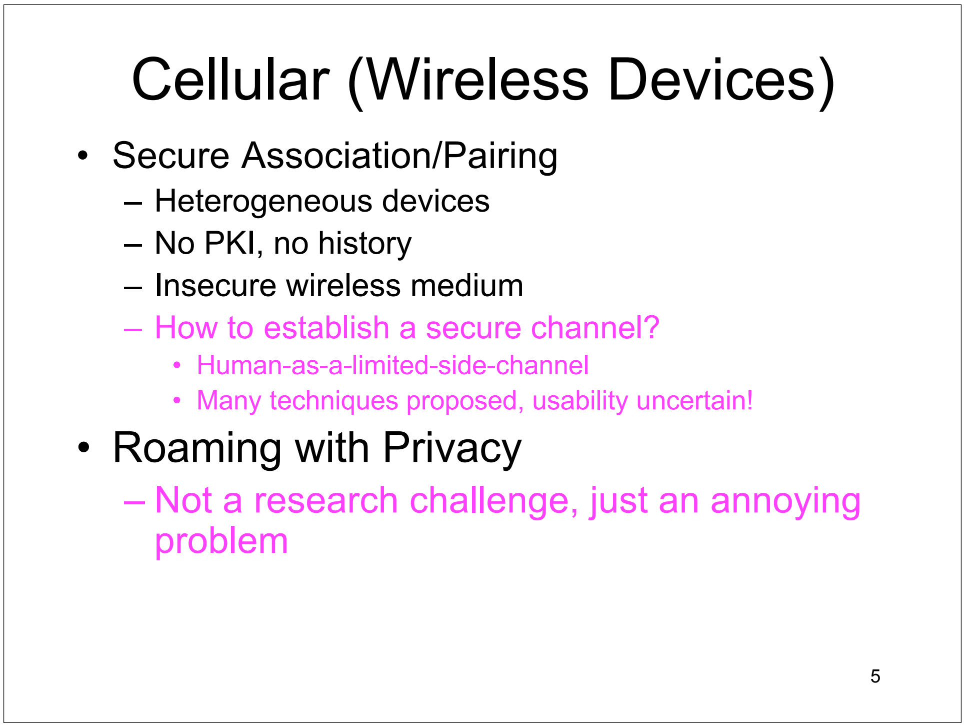
WSNs

- Survivability & Intrusion Resilience
  - Not all sensors network; many don't
  - Collect data, wait for pickup
  - Irregular pickups + possibly long intervals
  - Hostile environment -- intrusion/compromise possible
  - How to protect collected data?
    - Storage, Computation and Bandwidth constraints
- Secure Scalable Initialization
  - Sensors manufactured in large quantities
  - Key pre-distribution not always viable
  - How to quickly and securely "pair" groups of sensors?
    - Scalability? Usability?
- Graceful Degradation
  - Strong security can kill sensors
  - When death is near, is strong security important?
  - How to gracefully degrade/relax security services?
  - What metrics to use when degrading?
- Whither RFID/Sensor hybrid?



MANETs

- Oblivious Routing
  - Fixed node population
  - Nodes move, topology changes
  - Hostile environment
  - How to protect topology (even from insiders)
    - But keep security
- Location-based Addressing
  - Nodes don't have identities
  - Node instances known by current location
  - Node instances must be authentic
  - How to keep node behavior (movements) private?
    - Most routing protocols can't hack it..



Cellular (Wireless Devices)

- Secure Association/Pairing
  - Heterogeneous devices
  - No PKI, no history
  - Insecure wireless medium
  - How to establish a secure channel?
    - Human-as-a-limited-side-channel
    - Many techniques proposed, usability uncertain!
- Roaming with Privacy
  - Not a research challenge, just an annoying problem

Session VI – Wireless Sensor Networks, MANET, and Cellular Security

Session Chair: Peng Ning

Diversify Sensor Nodes to Improve Security of Sensor Networks

Wenliang Du
Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, NY 13244-1240 USA

A **fundamental challenge** in securing sensor networks is that sensor nodes can be physically compromised. Most of the security mechanisms relies on the secrecy of some important data that is stored on sensor nodes. For example, for encryption, the security depends on the secrecy of keys. Because of the lack of physical security and memory protection, sensors can be captured by adversaries, and secret keys stored in memories can be compromised. Once those secrets are disclosed, a sensor is *completely* compromised, i.e., adversaries can command the sensor to behave maliciously. It is important to protect those sensitive data even if sensor nodes are compromised.

Our goal is not restricted to protect each node, but instead, to protect a significant number of sensors from being compromised. To avoid failure caused by a few malfunctioned or malicious sensors, sensor-network applications often adopt fault-tolerance technologies, so the compromise of a small number of sensor nodes does not compromise the entire mission. However, when a significant number of sensors are compromised, the trusted computing infrastructure depended upon by sensor networks can be compromised.

Challenges

Challenge 1: Disguising Sensitive Data. Secret data are normally stored in memories, so once adversaries have understood the memory layout, they can easily retrieve the sensitive data by dumping the entire memory. To defeat such naive memory-dumping attacks, these data need to be disguised, so knowing the memory layout alone cannot find the data. Adversaries must also understand the program in order to find out where each the data are stored. The challenges is how to automate such data disguising process.

Challenge 2: Obfuscating Code With the modern code analysis, debugging, and reverse engineering tools, adversaries can understand the program and then find the sensitive data. It is essential to make code understanding difficult. Code obfuscation has been extensively studied for traditional systems; its main goal is to increase the complexity of code to defeat reverse engineering efforts. During the past, a number of interesting techniques have been developed in the literature. However, these techniques were developed for traditional systems with abundant power and resources. It is quite challenging to directly adopt them for sensor networks and embedded systems.

Challenge 3: Diversifying Code Code obfuscation is not foolproof; dedicated adversaries can eventually get the confidential data from a captured node. Although it might take adversaries quite a significant amount of time to succeed, if the programs running on different sensors are the same or similar, once a node is compromised, compromising another node takes much less time. We need to use *diversity* techniques to turn the same piece of software into many diversified versions, such that a comparative study of an already-compromised node and a newly-captured node is still difficult. A great challenge is how to diversity code to defeat both static and dynamic matching attacks, without consuming too much resources.

Innovations

Due to the energy constraints of sensor networks, any viable disguising, obfuscation, and diversification method should be energy efficient. This will lead to studies and innovations that are significantly different from the traditional code obfuscation. Moreover, sensor networks and embedded systems have their own unique properties, some of which might benefit code obfuscation.

Code diversification has been used extensively to provide robustness to systems; however there is not much study in using it to enhance security. Robustness mostly deals with accidental fault, but for security, we face intelligent adversaries with sophisticated tools. Therefore, this research can lead to innovative technologies that can be applied to not only sensor networks, but also embedded systems.

Detection of Abnormalities in MANETs

Wenye Wang, ECE Department, NC State University

1 Fundamental Limitations of Today's Solutions

Abnormalities in MANETs can be malicious attacks or selfish nodes which can affect network architecture and network operation significantly.

1.1 No Detection of Abnormalities

Without detection of abnormalities, secure routing can be considered as a *proactive* solution. Recently many secure routing protocols, such as ARAN, Adriadne, SAODV, SRP, SEAD, have been proposed to protect multihop wireless networks from malicious attacks that interrupt routing or [1, 2]. Clearly, there are two *distinct* objectives:

- Security is a goal: In this category, the idea is to show how attacks against ad-hoc and sensor networks, and analyze the security of all routing protocols. The objective is to design/examine attacks and develop countermeasures [3].
- Routing is a goal: The objective of these works is to design/modify current routing protocols, but adding new security features to prevent the routing from attacks and interruption.

For both directions, security analysis has been addressed along with peer-to-peer networking architecture for MANETs and sensor networks. In short, there are 10 attacks addressed by most of these works, except each work discusses one or more specific attacks that are not covered by others: *spoofing of IP address, forging of route request, forging of route reply, injecting route reply without receiving a route request, replay attack, rushing attacks, generating false errors, jamming, man-in-the-middle attack, modifying node list on a route request.* Almost all of these works are based on simulations and qualitative explanation without implementations in MANETs. An intuitive question is whether these solutions, or *at least* one solution is feasible to MANETs. To the best of our knowledge, NIST (National Institute of Standard and Technology) and UMBC developed the open source code of SAODV, which is also called SecAODV with IPv6. We found a technical report of their implementation with very few testing results [4]. In order to understand the functionalities of SecAODV, we used the open source code available at NIST and implemented on our testbed. Surprising, we found that the packet looses are in between 90%-100%! This simply tells us: a protocol could be very secure (from analysis), but might not be able to delivery data. The reasons for such a result are not fully explored which maybe one or combined factors, such as bugs in the code, optimization problem, or protocol design. However, it advises us how to make a secure protocol feasible in real systems.

1.2 With Detection of Abnormalities

On the other hand, there are many solutions that aim to design networks and networking protocols based upon the detection of abnormalities, which is more or less a *reactive* approach. In general, these solutions are designed to be adaptive to any threats or abnormalities in the network. The solutions to this end can be classified as

- Statistical methods: The main idea is to let each node (e.g., sensor nodes) to compute a statistical digest of the monitored phenomenon over a moving window of recent readings. By utilizing the statistical digests to aid in decision making and data aggregation. Wireless nodes may be set in promiscuous mode by overhearing others' broadcast message. The results of statistical digests are then used as a *trust* measure for path selection or topology control.

For example, to measure the node's cooperativeness, it is possible to study the characteristics of misbehaving nodes on the network layer. Selfish nodes, for the sake of saving energy, usually refuse to forward data packets for other nodes. Malicious nodes may intentionally drop partial data packets in a random or periodic manner. A malicious node may also pretend to be adjacent to a node actually faraway from it, thus trap all packets destined to that node afterward. Thus, dropping "transient" packets is one of the most common characteristics of misbehaviors.

- Empirical benchmark: The main idea is to use empirical benchmark, represented by stochastic models or trace files. Currently, there is almost nothing existed for mobile ad hoc networks, even small-scale experiments [5]. Although many new attacks are proposed, the security effectiveness against these attacks remain at the level of discussions and security analysis, even not present in simulations for most of the work. This brings a lot of arguments in the course of justification.

2 Research Challenges

- *Threat models*: Wireless or sensor nodes may be compromised or physically captured. Adversaries can control the compromised nodes and gain access to secret information stored in them. Thus, they can launch multiple attacks like dropping or altering the message contents going through them, so as to prevent the sink from receiving authentic sensor readings. Also, there may be colluded attacks where two or more nodes collaborate to let the false reports escape detection in the downstream path to the sink.
- *Measurements and computation*: Once threats models are defined, the subsequent issue is how to measure or detect threats according to the threats models and the cost at which these measurements are collected and processed.
- *Performance*: While in the design of security solutions (network) performance might not be a focus, it is necessary to ensure that a security-oriented algorithm or protocol can be incorporated into a networking protocol without making severe performance degradation. This is a very challenging issue for detection of abnormalities which often time relies on a long-term observation.

Ideally, a powerful detection tool, similar to intrusion detection for the Internet, is expected.

References

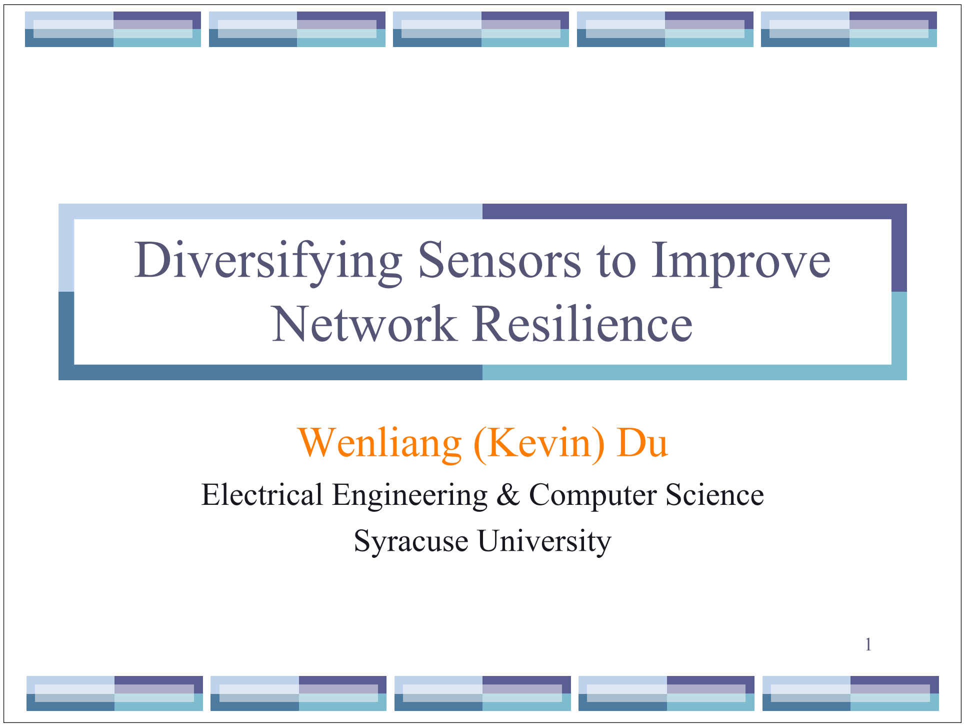
[1] P. Lee, V. Misra, and D. Rubenstein, "Distributed Algorithm for Secure Multipath Routing," in *Proc. of IEEE INFOCOM'05*, March 2005.

[2] K. Sanzgir, D. LaFlamme, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer, "Authenticated Routing for Ad Hoc Networks," *To Appear in IEEE Journal on Selected Areas of Communications (JSAC)*, 2005.

[3] C. Karlof and D. Wagner, "Secure Routing in Sensor Networks: Attacks and Countermeasures," *To Appear in Elsevier's AdHoc Networks Journal, Special Issue on Sensor Network Applications and Protocols.*, 2006.

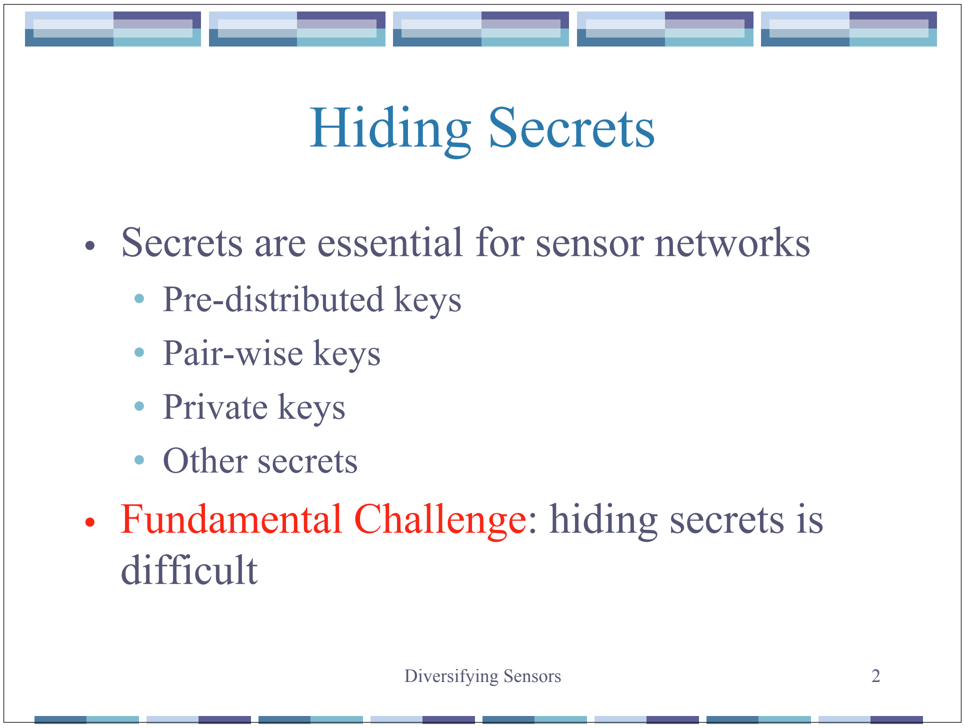
[4] A. Patwardhan, J. Parker, A. Joshi, M. Iorga, and T. Karygiannis, "Secure Routing and Intrusion Detection in Ad Hoc Networks," tech. rep., UMBC and NIST, 2003.

[5] D. M. Nicol, W. H. Sanders, and K. S. Trivedi, "Model-Based Evaluation: From Dependability to Security," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 48–65, 2004.



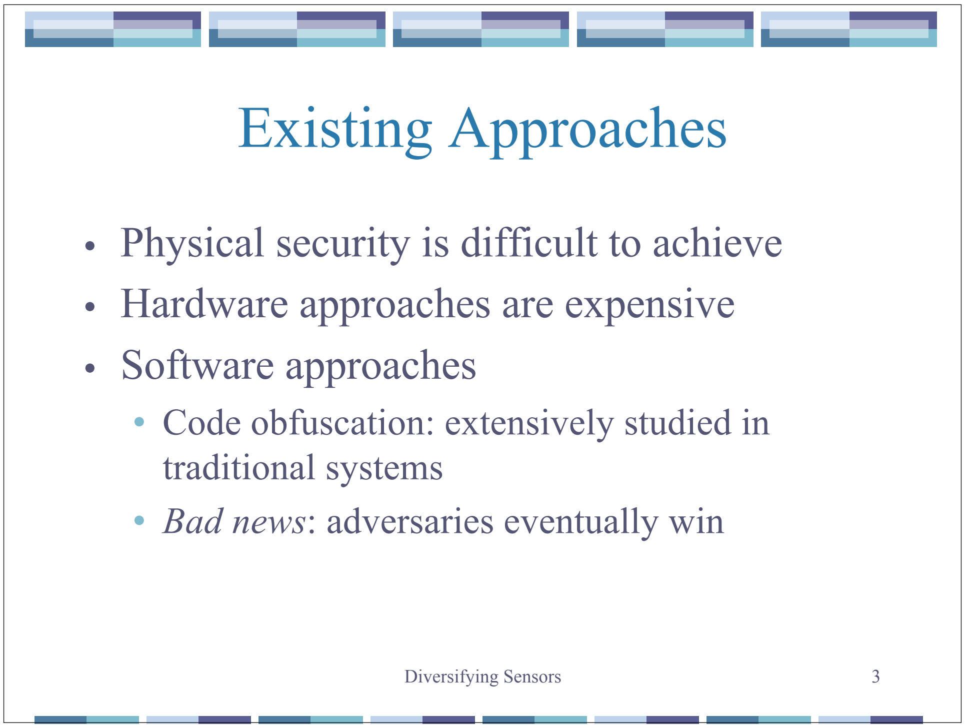
Diversifying Sensors to Improve Network Resilience

Wenliang (Kevin) Du
Electrical Engineering & Computer Science
Syracuse University



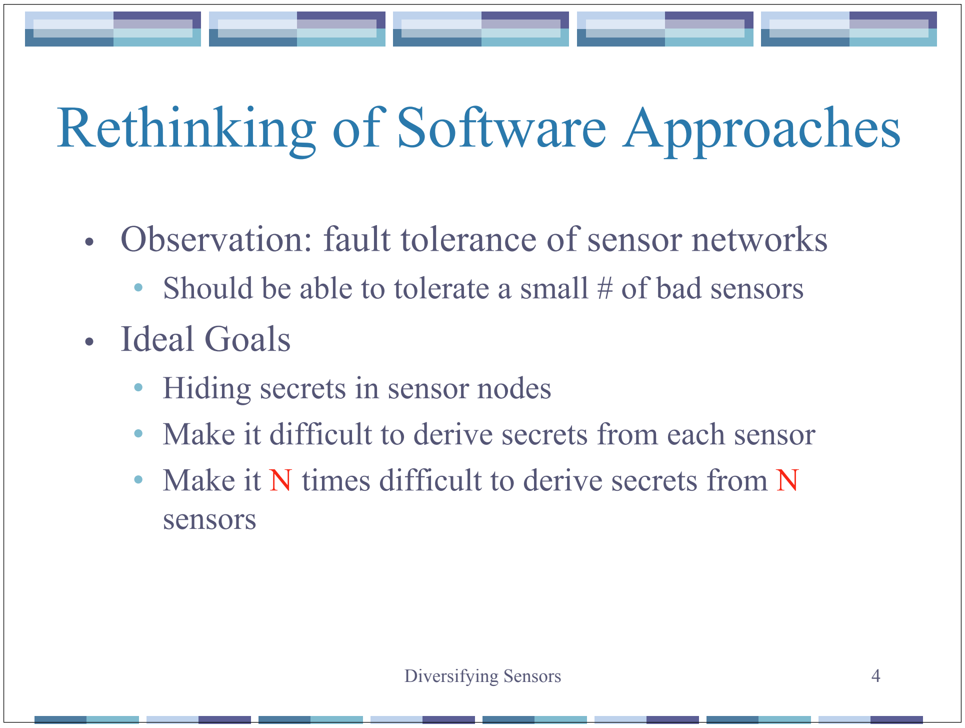
Hiding Secrets

- Secrets are essential for sensor networks
  - Pre-distributed keys
  - Pair-wise keys
  - Private keys
  - Other secrets
- Fundamental Challenge: hiding secrets is difficult



Existing Approaches

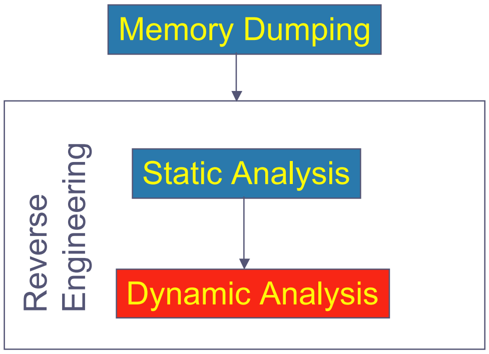
- Physical security is difficult to achieve
- Hardware approaches are expensive
- Software approaches
  - Code obfuscation: extensively studied in traditional systems
  - *Bad news*: adversaries eventually win



Rethinking of Software Approaches

- Observation: fault tolerance of sensor networks
  - Should be able to tolerate a small # of bad sensors
- Ideal Goals
  - Hiding secrets in sensor nodes
  - Make it difficult to derive secrets from each sensor
  - Make it N times difficult to derive secrets from N sensors

Threat Model: Physical Compromise



Proposed Approach

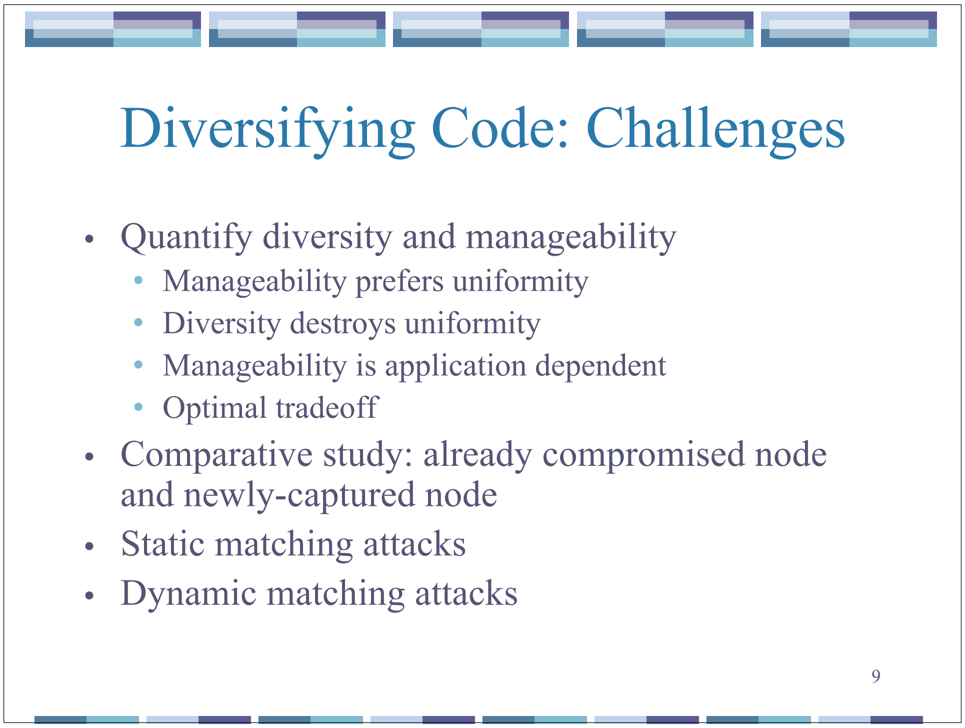
- Data Obfuscation (Secret Hiding)
  - Memory dump: difficult to find secrets
  - Adversaries must understand the program
- Code Obfuscation
  - Make it difficult to understand one program
- Code Diversification (Randomization)
  - Make adversary’s effort non-repeatable

Data/Code Obfuscation

- Existing Techniques
  - Code flattening
  - Self-modification code
  - White-box encryption algorithms
  - Various techniques against reverse engineering
- Challenges
  - Achieving obfuscation with limited Memory
  - Computation can’t be too expensive
  - Tradeoff needs to be made (optimization)
  - Quantify code complexity

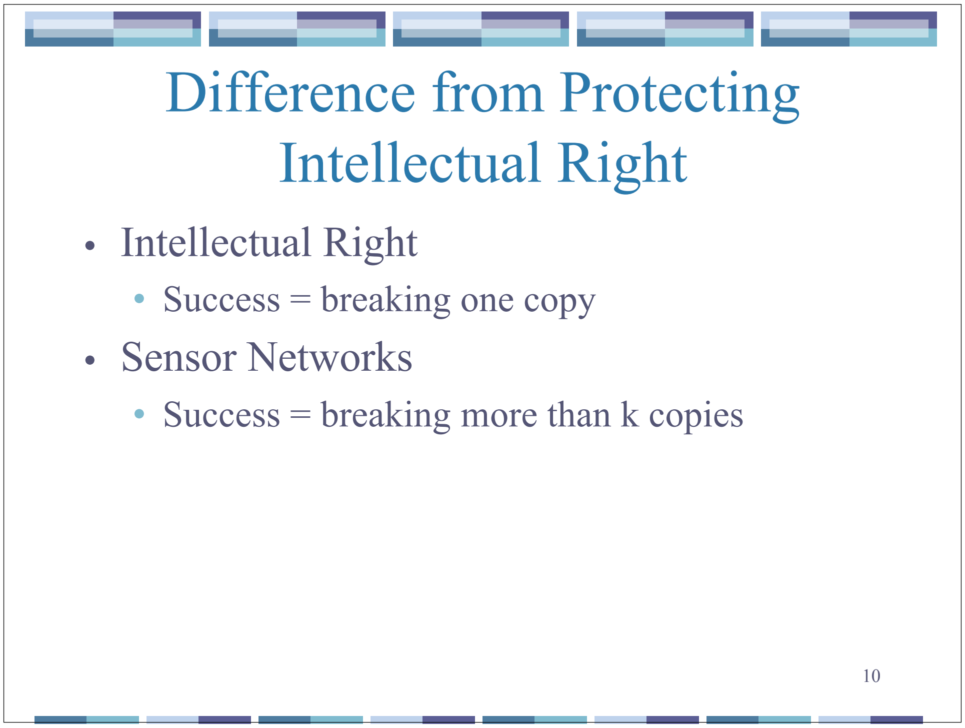
Diversifying Code

- Turn the same piece of software into many diversified versions
- Difference from traditional diversity
  - Diversity for fault tolerance
  - Diversity for attack tolerance (vulnerabilities)
    - Attacks are quite fragile
  - Diversity for code-analysis tolerance
    - Attacks are adaptive and intelligent (human involved)



Diversifying Code: Challenges

- Quantify diversity and manageability
  - Manageability prefers uniformity
  - Diversity destroys uniformity
  - Manageability is application dependent
  - Optimal tradeoff
- Comparative study: already compromised node and newly-captured node
- Static matching attacks
- Dynamic matching attacks



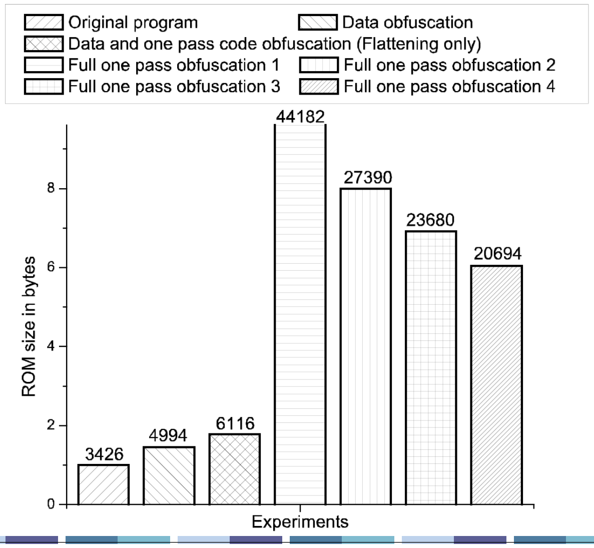
Difference from Protecting Intellectual Right

- Intellectual Right
  - Success = breaking one copy
- Sensor Networks
  - Success = breaking more than k copies

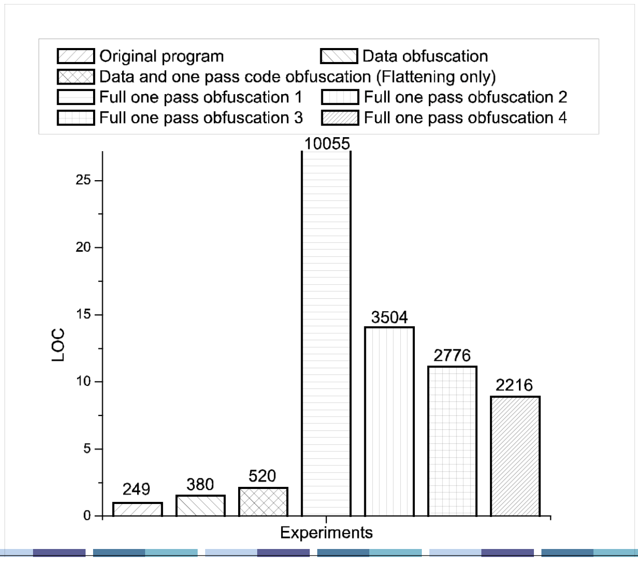
Unique Properties of Sensor Networks

- Code usually has small size
- Some applications has static configurations
  - The OS can be obfuscated too
- Hardware specific code obfuscation

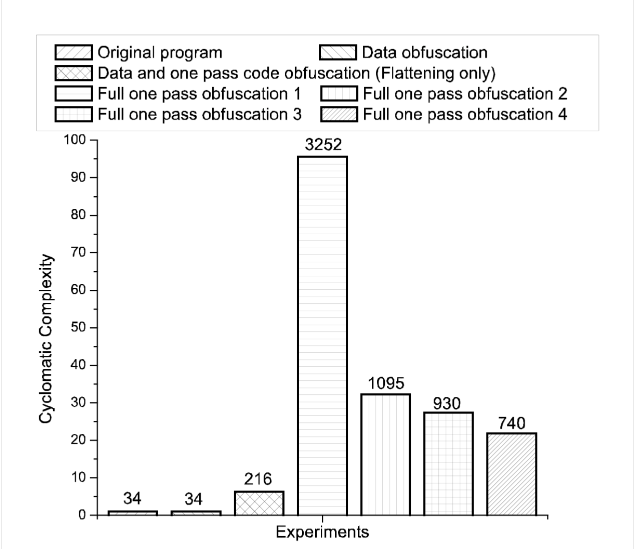
Preliminary Results: SASN'06

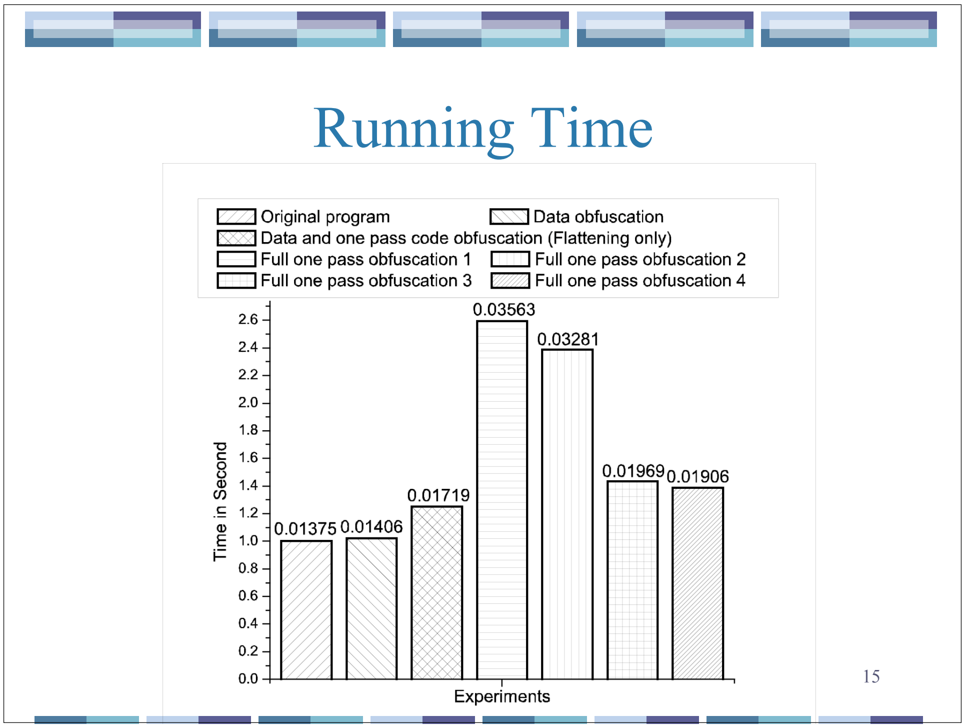


Complexity: Line of Code

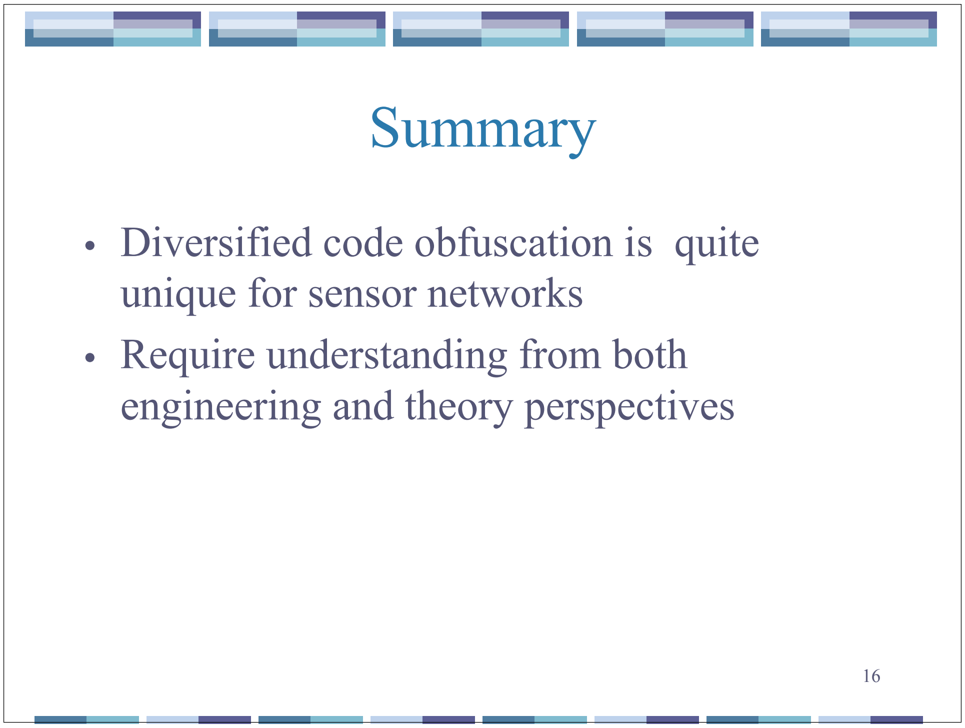


Cyclomatic Complexity





Running Time



Summary

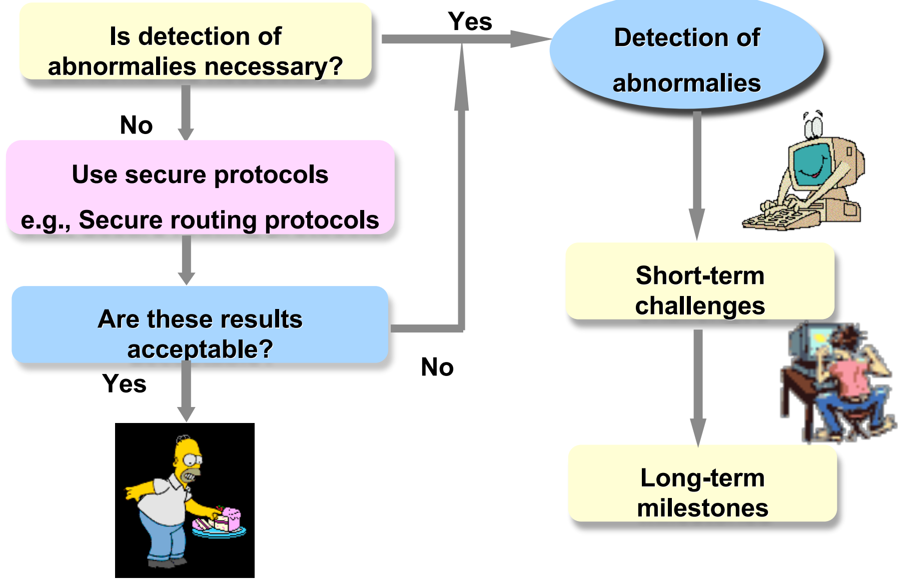
- Diversified code obfuscation is quite unique for sensor networks
- Require understanding from both engineering and theory perspectives

Detection of Abnormalies in MANETs

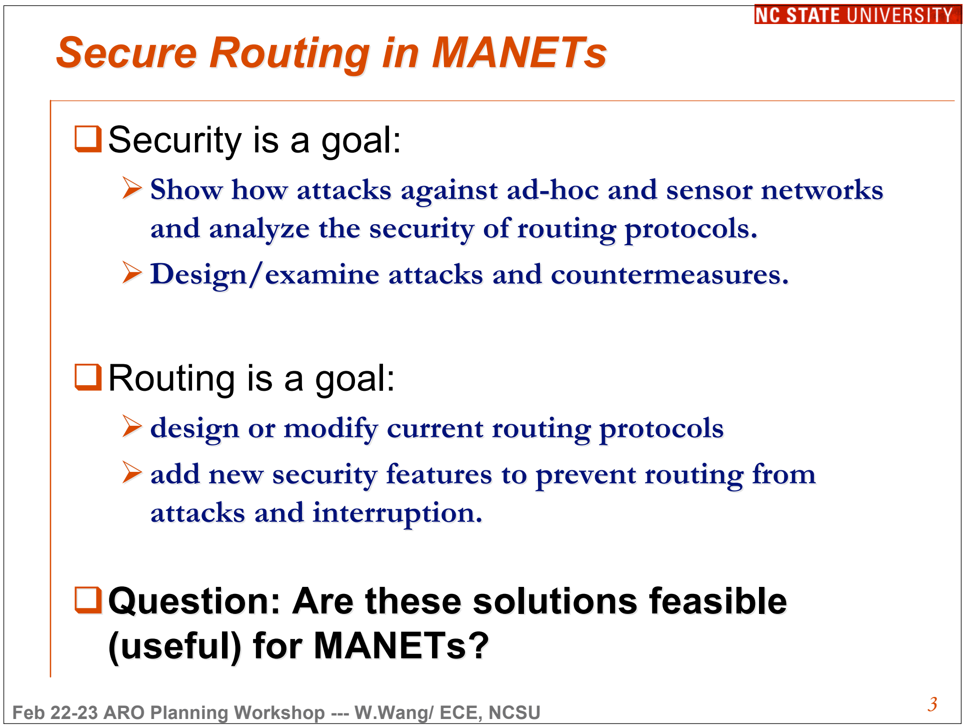
Wenye Wang

Department of Electrical and Computer Engineering
North Carolina State University

From Problems To Solutions

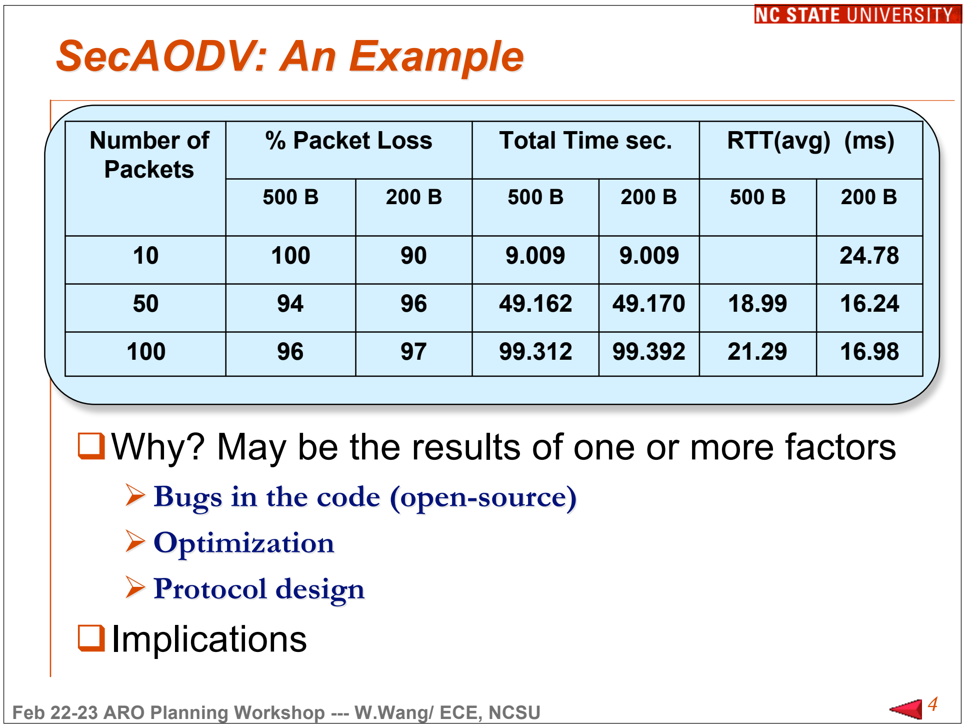

Is detection of abnormalies necessary?
Yes
Detection of abnormalies
No
Use secure protocols
e.g., Secure routing protocols
Are these results acceptable?
No
Yes
Short-term challenges
Long-term milestones

END



Secure Routing in MANETs

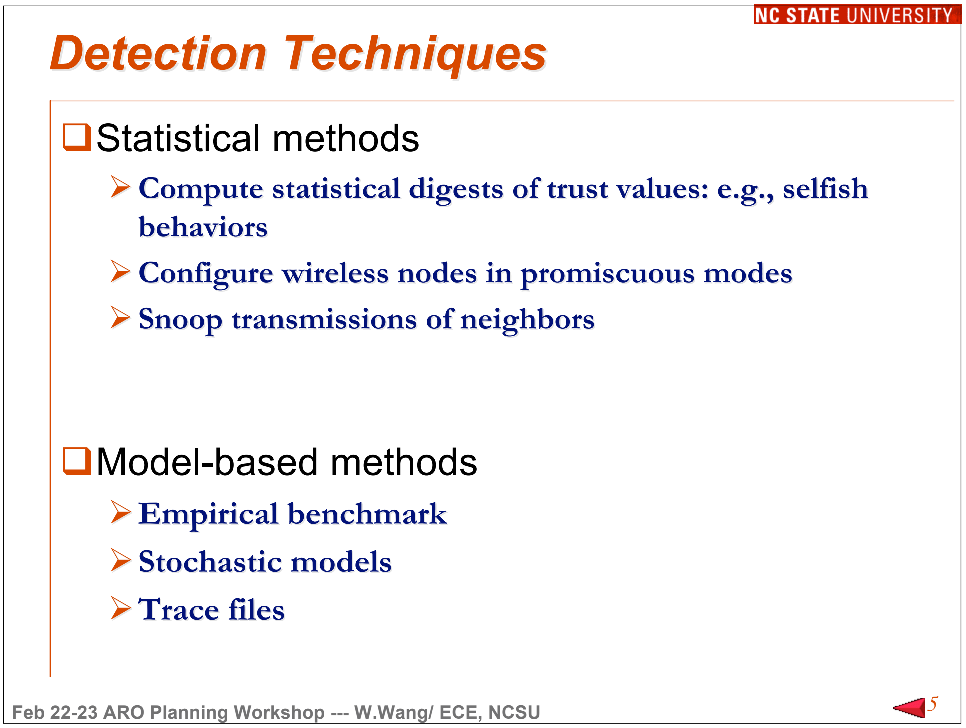
- ❑ Security is a goal:
  - ➢ Show how attacks against ad-hoc and sensor networks and analyze the security of routing protocols.
  - ➢ Design/examine attacks and countermeasures.
- ❑ Routing is a goal:
  - ➢ design or modify current routing protocols
  - ➢ add new security features to prevent routing from attacks and interruption.
- ❑ **Question: Are these solutions feasible (useful) for MANETs?**



SecAODV: An Example

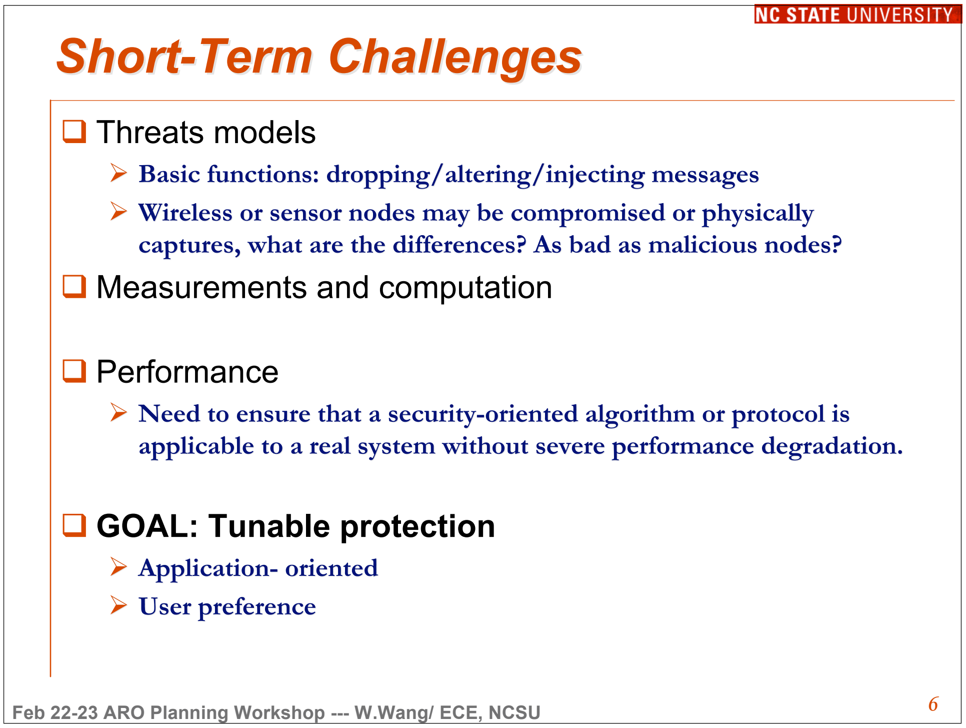
Number of Packets	% Packet Loss		Total Time sec.		RTT(avg) (ms)	
	500 B	200 B	500 B	200 B	500 B	200 B
10	100	90	9.009	9.009		24.78
50	94	96	49.162	49.170	18.99	16.24
100	96	97	99.312	99.392	21.29	16.98

- ❑ Why? May be the results of one or more factors
  - ➢ Bugs in the code (open-source)
  - ➢ Optimization
  - ➢ Protocol design
- ❑ Implications



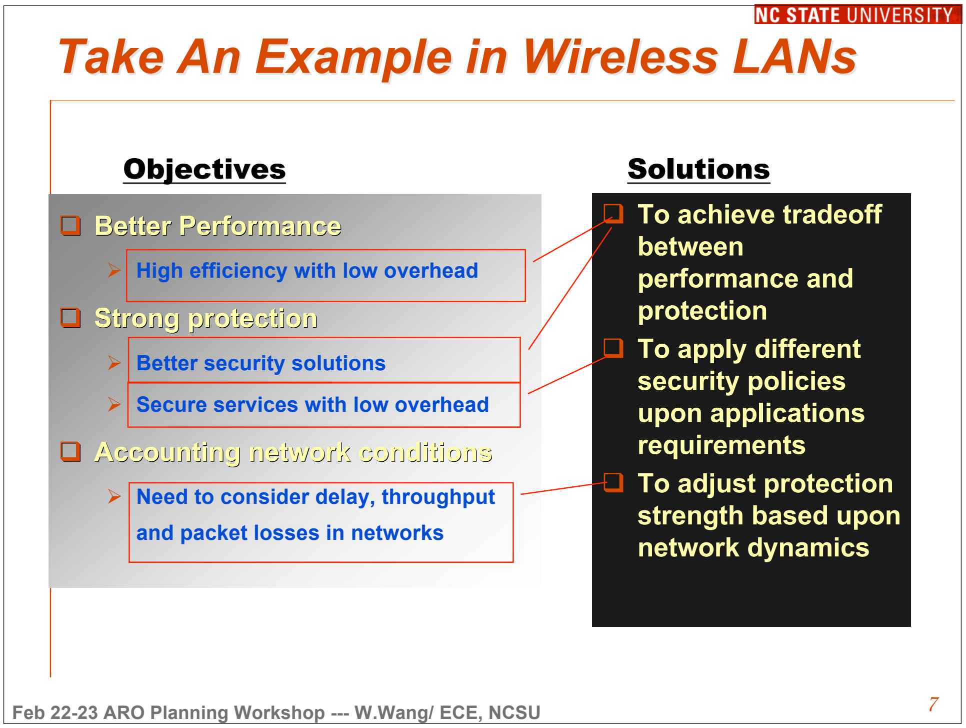
Detection Techniques

- ❑ Statistical methods
  - ➢ Compute statistical digests of trust values: e.g., selfish behaviors
  - ➢ Configure wireless nodes in promiscuous modes
  - ➢ Snoop transmissions of neighbors
- ❑ Model-based methods
  - ➢ Empirical benchmark
  - ➢ Stochastic models
  - ➢ Trace files



Short-Term Challenges

- ❑ Threats models
  - ➢ Basic functions: dropping/altering/injecting messages
  - ➢ Wireless or sensor nodes may be compromised or physically captures, what are the differences? As bad as malicious nodes?
- ❑ Measurements and computation
- ❑ Performance
  - ➢ Need to ensure that a security-oriented algorithm or protocol is applicable to a real system without severe performance degradation.
- ❑ **GOAL: Tunable protection**
  - ➢ Application- oriented
  - ➢ User preference



NC STATE UNIVERSITY

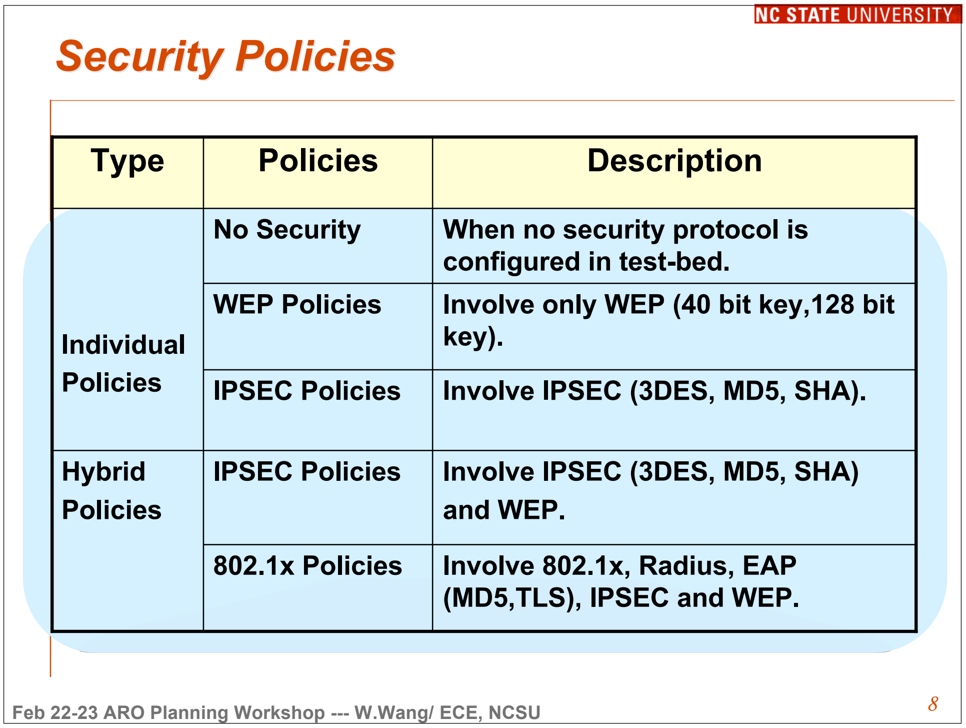
Take An Example in Wireless LANs

Objectives

- ❑ Better Performance
  - ➢ High efficiency with low overhead
- ❑ Strong protection
  - ➢ Better security solutions
  - ➢ Secure services with low overhead
- ❑ Accounting network conditions
  - ➢ Need to consider delay, throughput and packet losses in networks

Solutions

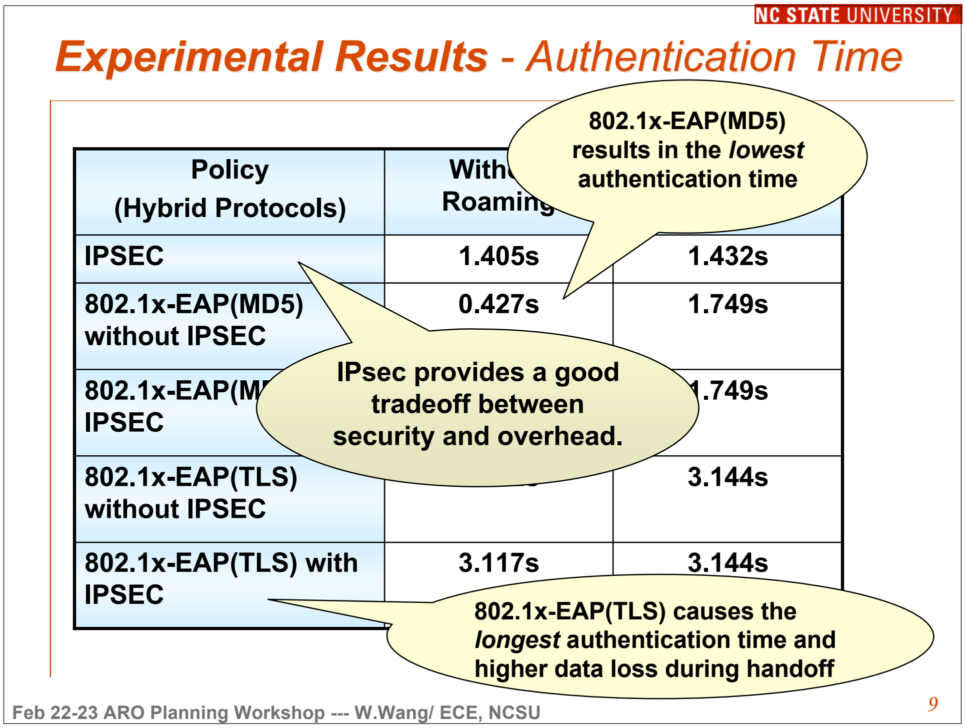
- ❑ To achieve tradeoff between performance and protection
- ❑ To apply different security policies upon applications requirements
- ❑ To adjust protection strength based upon network dynamics



NC STATE UNIVERSITY

Security Policies

Type	Policies	Description
Individual Policies	No Security	When no security protocol is configured in test-bed.
	WEP Policies	Involve only WEP (40 bit key,128 bit key).
	IPSEC Policies	Involve IPSEC (3DES, MD5, SHA).
Hybrid Policies	IPSEC Policies	Involve IPSEC (3DES, MD5, SHA) and WEP.
	802.1x Policies	Involve 802.1x, Radius, EAP (MD5,TLS), IPSEC and WEP.



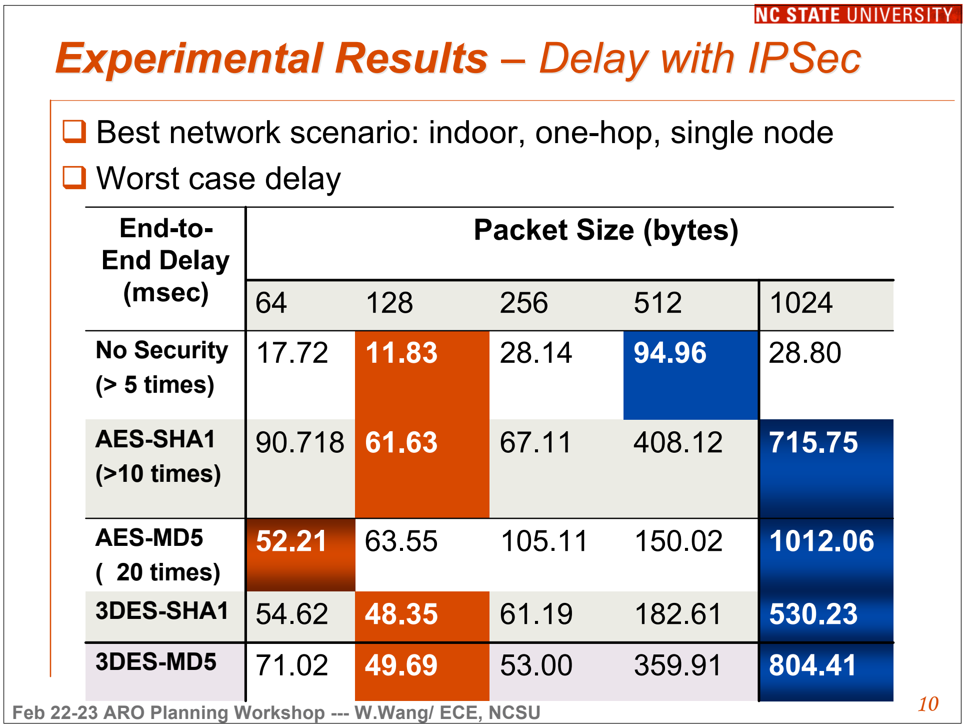
Experimental Results - *Authentication Time*

Policy (Hybrid Protocols)	With Roaming	
IPSEC	1.405s	1.432s
802.1x-EAP(MD5) without IPSEC	0.427s	1.749s
802.1x-EAP(M IPSEC		1.749s
802.1x-EAP(TLS) without IPSEC		3.144s
802.1x-EAP(TLS) with IPSEC	3.117s	3.144s

802.1x-EAP(MD5) results in the *lowest* authentication time

IPsec provides a good tradeoff between security and overhead.

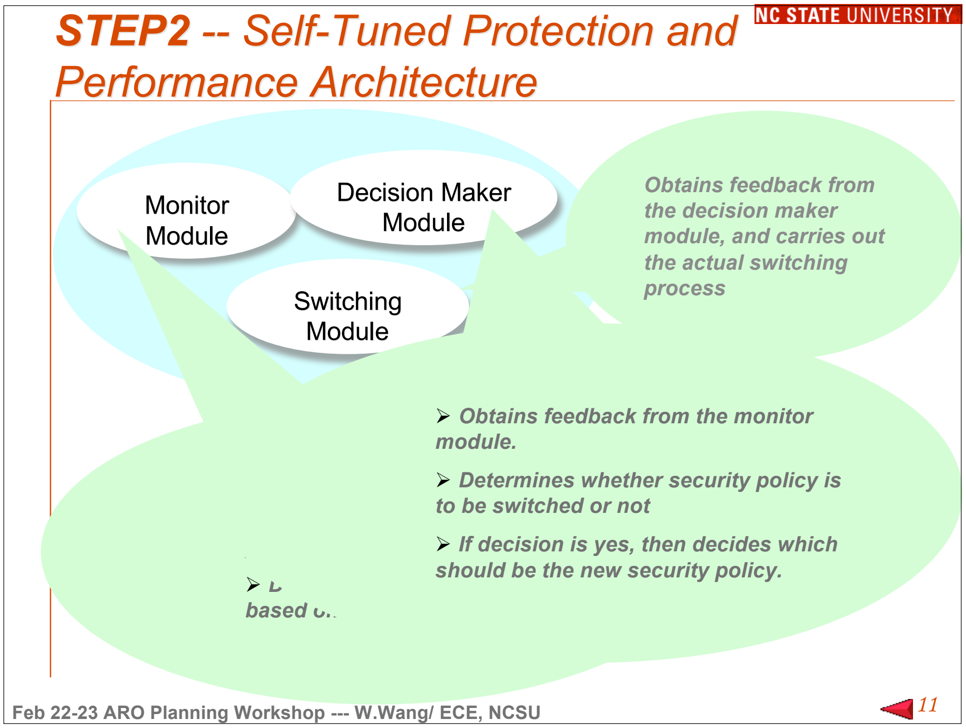
802.1x-EAP(TLS) causes the *longest* authentication time and higher data loss during handoff



Experimental Results – *Delay with IPSec*

- ❑ Best network scenario: indoor, one-hop, single node
- ❑ Worst case delay

End-to-End Delay (msec)	Packet Size (bytes)				
	64	128	256	512	1024
No Security (> 5 times)	17.72	**11.83**	28.14	**94.96**	28.80
AES-SHA1 (>10 times)	90.718	**61.63**	67.11	408.12	**715.75**
AES-MD5 (20 times)	**52.21**	63.55	105.11	150.02	**1012.06**
3DES-SHA1	54.62	**48.35**	61.19	182.61	**530.23**
3DES-MD5	71.02	**49.69**	53.00	359.91	**804.41**

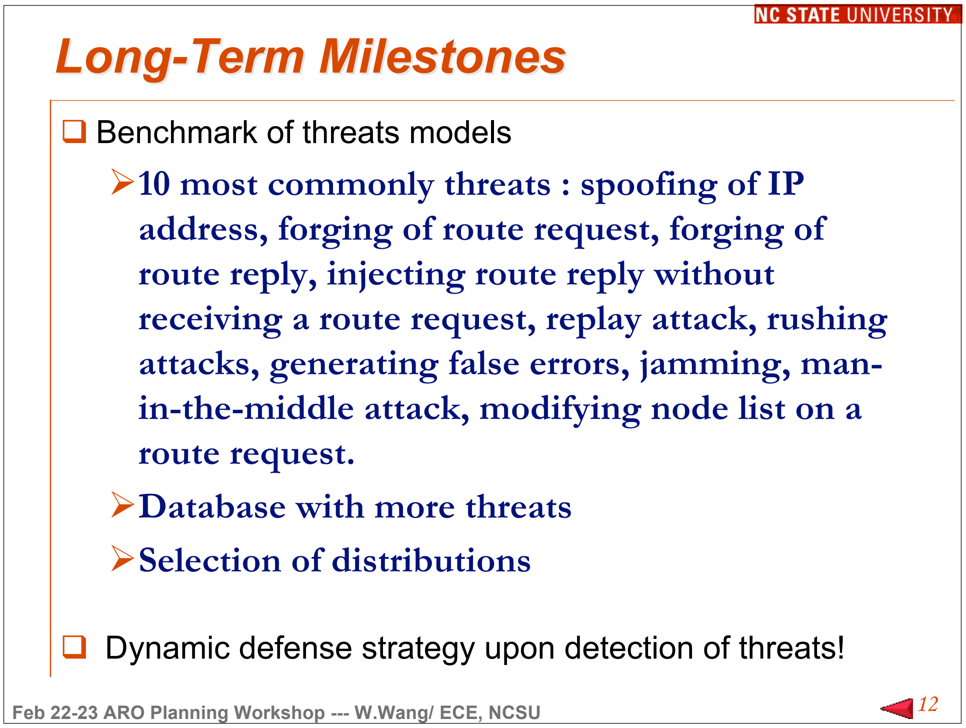


STEP2 -- Self-Tuned Protection and Performance Architecture

Obtains feedback from the decision maker module, and carries out the actual switching process

- Obtains feedback from the monitor module.
- Determines whether security policy is to be switched or not
- If decision is yes, then decides which should be the new security policy.

- D based o

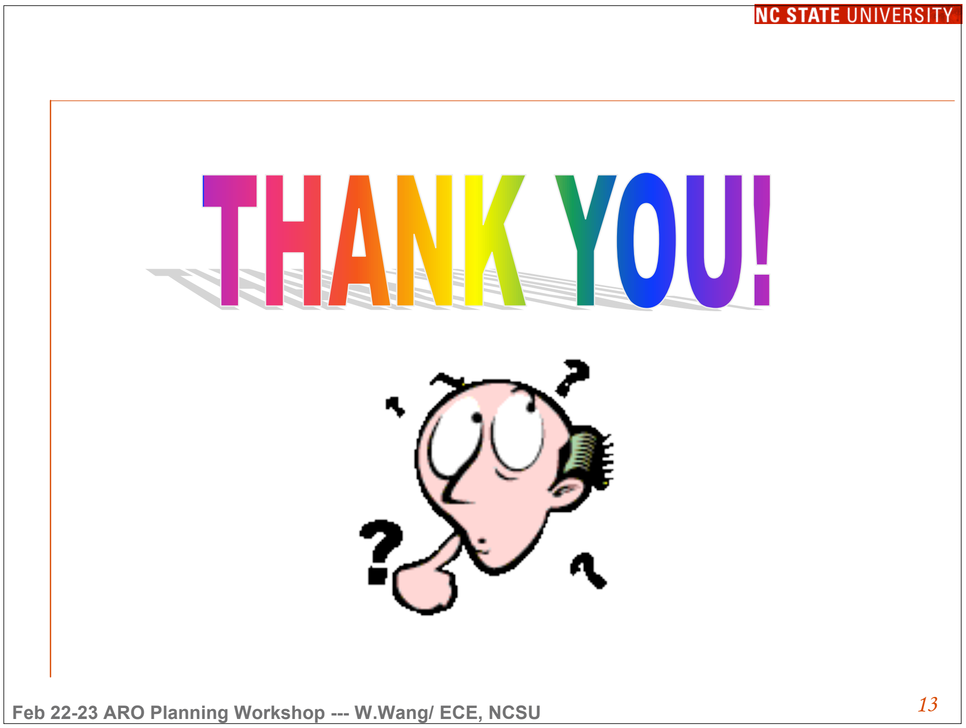


Long-Term Milestones

❑ Benchmark of threats models

- 10 most commonly threats : spoofing of IP address, forging of route request, forging of route reply, injecting route reply without receiving a route request, replay attack, rushing attacks, generating false errors, jamming, man-in-the-middle attack, modifying node list on a route request.
- Database with more threats
- Selection of distributions

❑ Dynamic defense strategy upon detection of threats!



THANK YOU!

Summary and Outlook – Research Challenges

Open Research Questions

- Adversary models
  - Define/Formalize adversary models
    - Need to incorporate characteristics of new technologies and applications
    - Need to consider the threats to the defense mechanisms
  - Define performance/security metrics
  - Develop process/methodology for developing adversary models

Languages and Software Engineering (1)

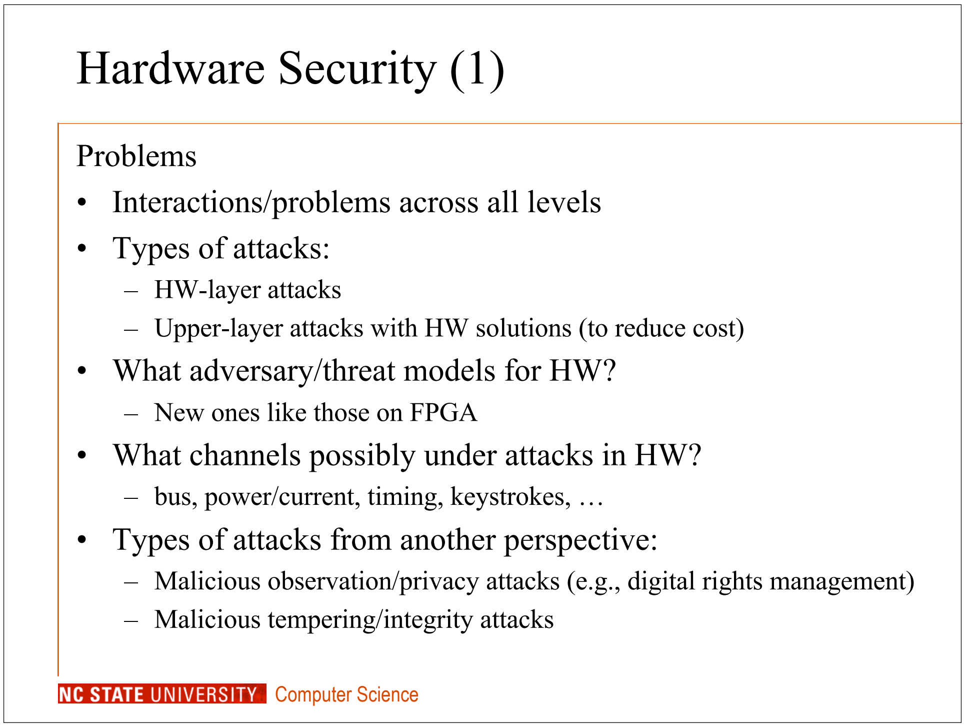
- Embedded constraints affect security
  - Quantify capabilities, limit scope, target 8-bit & larger
  - beyond sensor nodes --> deeply embedded systems
  - Issues: critical, non-recoverable, special netw., no sys admin
- Design for security (not retrofit) → formalize
- Need for metrics & models (some differ for embedded)
  - e.g. higher reliability probability for safety crit. affects security
- Need for diversity (e.g., via dynamic adaptation)
  - Self-modifying apps (e.g., via dyn. Transformation of pgm)
  - Self-protecting/self-checking apps?
  - Dynamic updates (in 24/7 operation)
  - Classification in terms of threat models

Languages and Software Engineering (2)

- Need to limit overhead, retain predictability, low cost
- Incorporate real-time requirements
  - Hard timing constraints limit security options
  - may undermine existing network protocols, add overhead…
  - Interface b/w RT and non-RT components problematic
  - Embedded clients + server need protection of both
- Need hardware assistance

Software Security

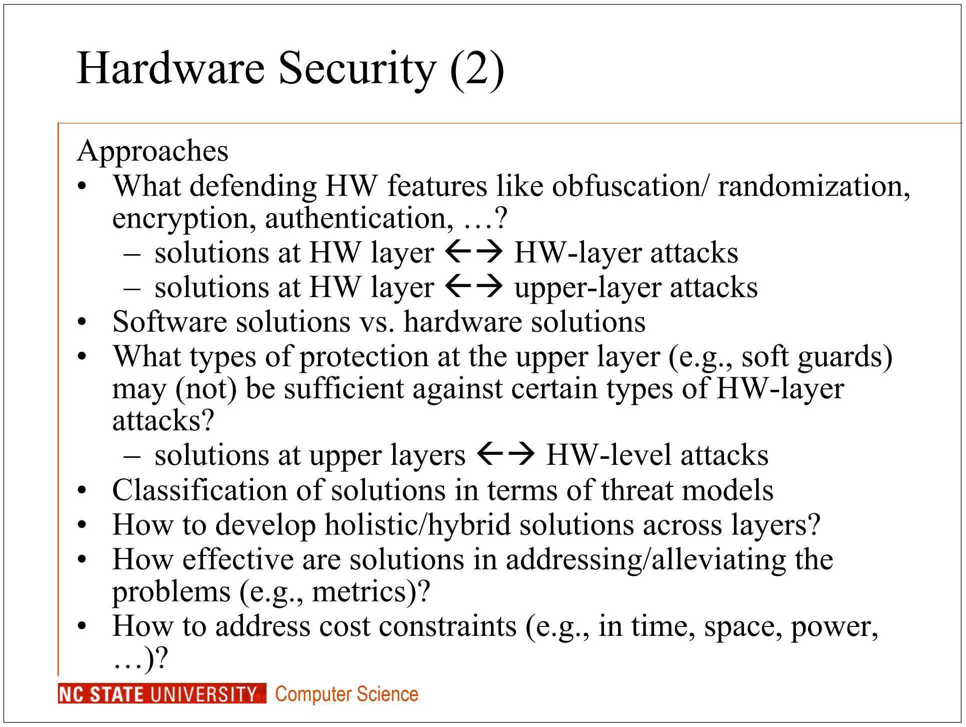
- Can light-weight, effective, semantics-based, compiler-level reasoning systems to characterize program behavior, mal-ware, etc be built?
- Can effective hybrid reasoning systems be built by combining static analysis and runtime monitoring systems?
- Can evaluation contexts be characterized to simplify and reduce efforts in reasoning about software artifacts?
- How can binaries be reasoned about without too many false alarms?
- Can binaries be instrumented to discover security threats and to degrade gracefully in the event of a security fault?
- How to characterize threats, vulnerabilities,…?
- How to build provably correct core components of OS against specific threat definitions?
- Code integrity to leverage to achieve system security.
- Secure and scalable software update on deployed systems.
- Tool support for code obfuscation.



Hardware Security (1)

Problems

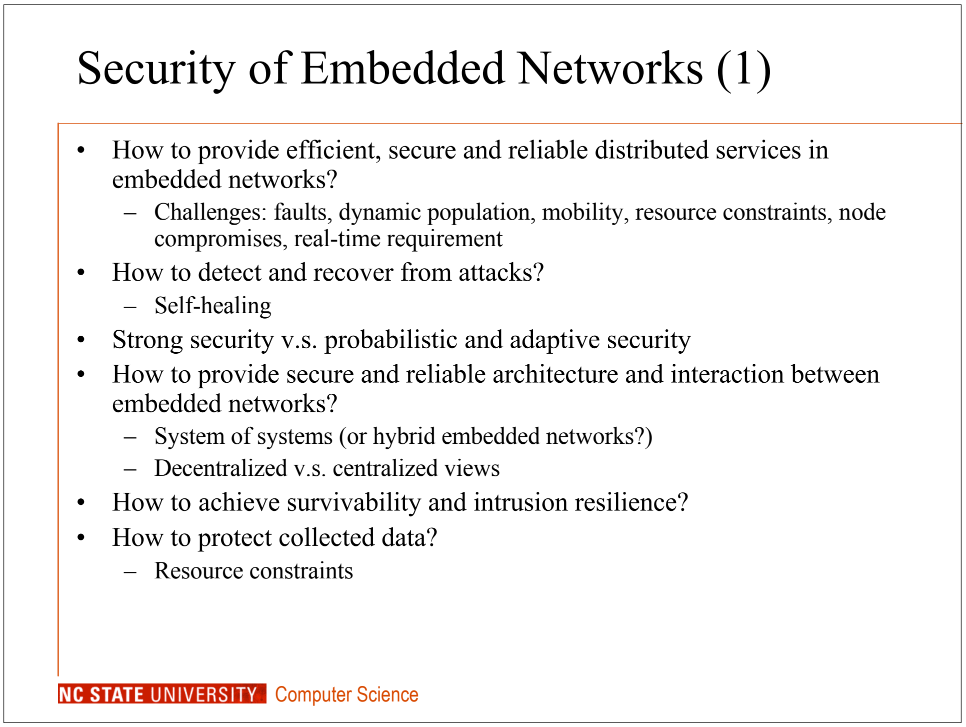
- Interactions/problems across all levels
- Types of attacks:
  - HW-layer attacks
  - Upper-layer attacks with HW solutions (to reduce cost)
- What adversary/threat models for HW?
  - New ones like those on FPGA
- What channels possibly under attacks in HW?
  - bus, power/current, timing, keystrokes, …
- Types of attacks from another perspective:
  - Malicious observation/privacy attacks (e.g., digital rights management)
  - Malicious tempering/integrity attacks



Hardware Security (2)

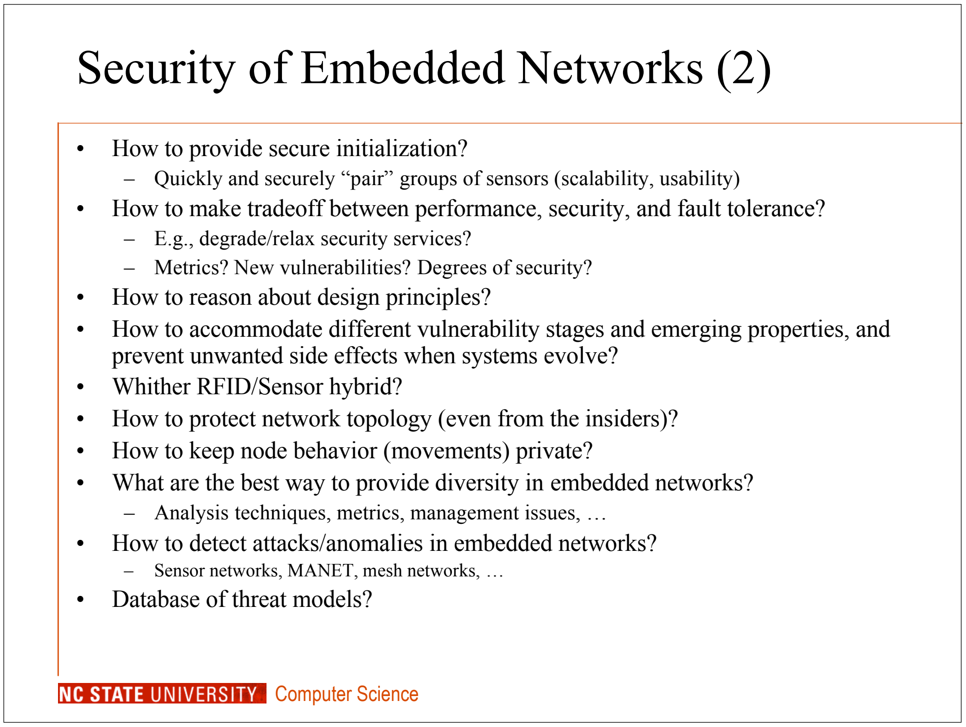
Approaches

- What defending HW features like obfuscation/ randomization, encryption, authentication, …?
  - solutions at HW layer ←→ HW-layer attacks
  - solutions at HW layer ←→ upper-layer attacks
- Software solutions vs. hardware solutions
- What types of protection at the upper layer (e.g., soft guards) may (not) be sufficient against certain types of HW-layer attacks?
  - solutions at upper layers ←→ HW-level attacks
- Classification of solutions in terms of threat models
- How to develop holistic/hybrid solutions across layers?
- How effective are solutions in addressing/alleviating the problems (e.g., metrics)?
- How to address cost constraints (e.g., in time, space, power, …)?



Security of Embedded Networks (1)

- How to provide efficient, secure and reliable distributed services in embedded networks?
  - Challenges: faults, dynamic population, mobility, resource constraints, node compromises, real-time requirement
- How to detect and recover from attacks?
  - Self-healing
- Strong security v.s. probabilistic and adaptive security
- How to provide secure and reliable architecture and interaction between embedded networks?
  - System of systems (or hybrid embedded networks?)
  - Decentralized v.s. centralized views
- How to achieve survivability and intrusion resilience?
- How to protect collected data?
  - Resource constraints



Security of Embedded Networks (2)

- How to provide secure initialization?
  - Quickly and securely “pair” groups of sensors (scalability, usability)
- How to make tradeoff between performance, security, and fault tolerance?
  - E.g., degrade/relax security services?
  - Metrics? New vulnerabilities? Degrees of security?
- How to reason about design principles?
- How to accommodate different vulnerability stages and emerging properties, and prevent unwanted side effects when systems evolve?
- Whither RFID/Sensor hybrid?
- How to protect network topology (even from the insiders)?
- How to keep node behavior (movements) private?
- What are the best way to provide diversity in embedded networks?
  - Analysis techniques, metrics, management issues, …
- How to detect attacks/anomalies in embedded networks?
  - Sensor networks, MANET, mesh networks, …
- Database of threat models?